JOÃO CARLOS ROCHA DE BORBA
VINICIUS DE CARVALHO HAIDAR

GERENCIADOR DE DIÁLOGOS PARA ROBÔ SOCIÁVEL

SÃO PAULO
2012

JOÃO CARLOS ROCHA DE BORBA
VINICIUS DE CARVALHO HAIDAR

GERENCIADOR DE DIÁLOGOS PARA ROBÔ SOCIÁVEL

Monografia apresentada ao Departamento de Engenharia Mecatrônica da Escola Politécnica da Universidade de São Paulo para obtenção do título de bacharel em Engenharia Mecatrônica.

SÃO PAULO
2012

JOÃO CARLOS ROCHA DE BORBA
VINICIUS DE CARVALHO HAIDAR

GERENCIADOR DE DIÁLOGOS PARA ROBÔ SOCIÁVEL

Monografia apresentada ao Departamento de Engenharia Mecatrônica da Escola Politécnica da Universidade de São Paulo para obtenção do título de bacharel em Engenharia Mecatrônica.

Área de concentração:
Engenharia Mecatrônica

Orientador: Prof. Dr. Marcos Ribeiro Pereira Barretto

SÃO PAULO
2012

FICHA CATALOGRÁFICA

Borba, João Carlos Rocha de
Gerenciador de diálogos para robô sociável / J.C.R.de Borba, V.C. Haidar. -- São Paulo, 2012.
95 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Robôs 2. Reconhecimento da fala 3. Transformação texto-fala I. Haidar, Vinicius de Carvalho II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos III. t.

DEDICATÓRIA

João: Dedico a meus pais, que com muito esforço me possibilitaram estudar e viver em São Paulo.

Vinicius: Dedico a minha família por todo o apoio e incentivo que sempre me deram em todos os momentos da minha vida.

AGRADECIMENTOS

Ao Prof. Dr. Marcos Pereira Barretto, pelas orientações técnicas e acadêmicas.

Ao Prof. Dr. Jun Okamoto Junior, que gentilmente nos permitiu utilizar as instalações do Laboratório de Percepção Avançada para o desenvolvimento do projeto.

Ao Eng. José Carlos dos Santos, pelo suporte técnico e pelas orientações técnicas no projeto e na montagem dos circuitos eletrônicos.

Aos demais membros do Laboratório de Percepção Avançada, por nos terem recebido gentilmente nas instalações e nos concederem a utilização das ferramentas disponíveis no recinto.

O tempo não tem significado real para mim.

(Minerva)

RESUMO

BORBA, J. C. R., HAIDAR, V. C. **Gerenciador de Diálogos para Robô Sociável**. 2012. Trabalho de Formatura – Departamento de Engenharia Mecatrônica, Escola Politécnica da Universidade de São Paulo, São Paulo, 2012.

Robôs sociáveis são robôs autônomos criados para interagir e se comunicar com seres humanos e outros agentes autônomos seguindo regras de comportamento social. Uma das principais habilidades destes robôs é se comunicar verbalmente, o que simplificadamente envolve três tarefas principais: ouvir, decidir por uma resposta e dizer a resposta. Este trabalho tem como objetivo projetar e desenvolver um gerenciador de diálogos para o robô sociável Minerva. O gerenciador é dividido em módulos: primeiro o módulo de reconhecimento de fala, responsável pela audição; em seguida um gerador de respostas em AIML, que gera a resposta do robô; por fim o módulo de transformação texto-fala, responsável por proferir a resposta. Há ainda o módulo construtor de prosódia, que adiciona aspectos prosódicos à fala do robô, como sonoridade interrogativa, visando tornar a fala mais verossímil à fala humana. Na integração do gerenciador de diálogos com o robô é inserido um último módulo, o sistema de movimentação para os lábios, que funciona de forma sincronizada com o discurso falado.

Palavras-chave: Robô Sociável. Gerenciador de Diálogos. Reconhecimento de Fala. Transformação Texto-Fala.

ABSTRACT

BORBA, J. C. R., HAIDAR, V. C. **Social Robot Dialog Manager**. 2012. Trabalho de Formatura – Departamento de Engenharia Mecatrônica, Escola Politécnica da Universidade de São Paulo, São Paulo, 2012.

Social robots are autonomous robots designed to interact and communicate to human beings and other autonomous agents according to social behavior rules. One of the main abilities of these robots is to use oral communication, what is composed by three different tasks: listen, decide for an answer, and say. This work presents the development of a dialog manager for a social robot. The system is divided in modules: speech recognition, responsible for listening; then an answer generator developed in AIML, which generates the robot's answer; and finally the text-to-speech module, responsible for pronouncing the response. There is still a prosody constructor module, which adds prosody aspects to the robot speech, such as interrogatives, seeking to turn the speech similar to a real human pronunciation. A last module is inserted in the integration of the dialog manager with the robot, the lips movement system, which synchronizes the mouth movements of the robot with the speech.

Keywords: Social Robot. Dialog Manager. Speech Recognition. Text-to-Speech.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

AIML *Artificial Intelligence Markup Language* - Linguagem Markup para Inteligência Artificial

ANN *Artificial Neural Network* – Rede Neural Artificial

API *Application Programming Interface* – Interface de Programação de Aplicativos

ASR *Automatic Speech Recognition* – Reconhecimento Automático de Fala

FTDI *Future Technology Devices International*

HMM *Hidden Markov Model* – Modelo Oculto de Markov

POS *Part-of-speech* – Parte do discurso

PWM *Pulse Width Modulation* – Modulação por Largura de Pulso

SNR *Signal to Noise Ratio* – Razão Sinal-Ruído

SSML *Speech Synthesis Markup Language* – Linguagem Markup para Síntese de Fala

TTS *Text-to-speech* – Transformação texto-fala

W3C *World Wide Web Consortium* – Consórcio de padronização da *World Wide Web*

XML *Extensible Markup Language* – Linguagem Markup Extensível

SUMÁRIO

1- INTRODUÇÃO

1.1- MOTIVAÇÃO

Robôs sociáveis são robôs autônomos que interagem e comunicam-se com humanos e outros agentes físicos autônomos seguindo as regras de comportamento social e as regras associadas ao seu papel na sociedade (ALFENAS E BARRETTO, 2012).

O robô Minerva é um robô sociável. Ele consiste em uma cabeça de mulher, na qual uma pele de resina esconde mecanismos que movimentam olhos, lábios e face. O projeto pretende que ela se pareça com um ser humano, tanto na forma, quanto no comportamento. Com Minerva, tenta-se avançar na pesquisa pelo desenvolvimento de robôs com comportamento similar ao humano. Algumas funções do robô estão implementadas, como mecanismos de movimentação do pescoço e dos olhos. Outras ainda precisam ser desenvolvidas, como expressões faciais, sistema de visão e memória.

Figura 1 – Robô Minerva

Robôs como Minerva podem ser utilizados como acompanhantes para doentes, idosos ou pessoas solitárias. Também podem ser utilizados como robôs que tenham que conhecer o interlocutor, ou seja, saber o que este faz, gosta ou possui. Alguns filmes futuristas já retrataram situações fictícias de convívio social entre humanos e robôs, dentre eles se destacam “O Homem Bicentenário”, de 1999, e “A.I. Inteligência Artificial”, de 2001. Ainda é possível identificar outras utilizações para robôs sociáveis, como: atendente de ponto de informações, recepcionista, vendedor, professor, contador de histórias, entre outras possíveis aplicações.

Os humanos interagem entre si de forma verbal e não verbal. A linguagem verbal dá-se através de palavras, enquanto que a não verbal envolve, por exemplo, transmissão de emoções pela voz ou por expressões da face. Na tentativa de permitir a uma máquina interagir com os humanos em sua linguagem natural, nos últimos anos os pesquisadores têm proposto arquiteturas de gerenciadores de diálogo. Esses gerenciadores são sistemas constituídos por módulos, onde cada módulo é responsável por tratar um aspecto do diálogo, como audição, interpretação, memória ou fala.

1.2- OBJETIVOS

Este trabalho tem como objetivo projetar e implementar um gerenciador de diálogos para o robô sociável Minerva. Será utilizada a arquitetura ilustrada na Figura 2, que é uma simplificação da proposta presente em (ALFENAS E BARRETTO, 2012). São mantidos os módulos principais: ASR (reconhecimento automático de fala), construtor de prosódia e TTS (transformação texto-fala). A simplificação está na construção do diálogo. Todos os módulos entre o ASR e o construtor de prosódia são substituídos por um único módulo, um *chatbot*[1] em AIML. Isso implica que não há interpretação nem memória, mas um sistema de correspondência direta entre entrada e saída. Isto é, para cada frase reconhecida, há um *template* de resposta correspondente, que é a resposta do robô.

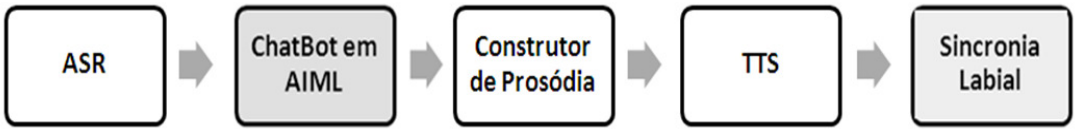

Figura 2 – Arquitetura simplificada do Gerenciador de Diálogos

O objetivo não é alcançar um diálogo natural complexo, mas sim diálogos simples, onde o robô consiga identificar o que está sendo dito e emita respostas que façam sentido. Deseja-se que este trabalho seja o ponto de partida para a implementação da arquitetura completa, bastando retirar o *chatbot* em AIML e acrescentar módulos de geração de diálogo com interpretação, memória e inclusão de emoções.

Nota-se ainda na Figura 2, que após o módulo TTS é incluído o quadro Sincronia Labial. Este módulo é a movimentação da boca do robô em sincronia com as consoantes e vogais que o mesmo pronuncia. Tal movimentação dá-se em um único grau de liberdade: a abertura e o fechamento da boca. A proposta é preparar a estrutura de software e hardware

[1] *Chatbot* é um programa de computador que tenta simular um ser humano na conversação com as pessoas. Seu objetivo é responder as perguntas de tal forma que as pessoas tenham a impressão de estar conversando com outra pessoa e não com um programa de computador. O termo será usado em inglês por falta de uma tradução adequada em língua portuguesa.

para uma sincronia labial com mais graus de liberdade, que poderá operar assim que os mecanismos dos lábios tenham sido desenvolvidos.

De forma sucinta, os objetivos do trabalho, separando por módulos, são:

a) ASR: Desenvolver o software de reconhecimento de fala com as API's do Loquendo ASR (LOQUENDO, 2011b), visando reconhecimento em vocabulários extensos, de forma a permitir a conversação com o robô.
b) *Chatbot* em AIML: Implementar um *chatbot* em AIML aplicável ao projeto. O *chatbot* deve conter categorias suficientes para gerar respostas que façam sentido e possam manter a conversação, mesmo que apenas durante algumas frases.
c) Construtor de Prosódia: Desenvolver no software de fala do robô o controle dos aspectos prosódicos (tom, intensidade, etc) do texto para tornar a fala mais parecida com a humana.
d) TTS: Utilizar as API's do software comercial Loquendo TTS (LOQUENDO, 2008) para desenvolver um aplicativo de transformação texto-fala.
e) Sincronia Labial: Projetar e montar o sistema de acionamento dos motores do robô e implementar o software de movimentação sincronizada da boca com a fala.
f) Integração: Integrar os aplicativos e a sincronização labial e testar a conversação do robô.

Os softwares Loquendo ASR e Loquendo TTS são conjuntos de API's (*Application Programming Interface*), ou seja, não são aplicativos prontos para reconhecimento de fala e transformação texto-fala, mas sim um conjunto de ferramentas que permitem criar tais aplicativos, conforme a necessidade. O uso de soluções comerciais para desenvolver esses módulos é a opção escolhida porque o desenvolvimento de soluções novas de ASR e TTS não faz parte do escopo do projeto.

Um dos desafios é identificar as potencialidades desses softwares e saber se eles são suficientemente eficazes para a tarefa que se pretende realizar. Para o ASR, por exemplo, fazer o reconhecimento correto das frases em vocabulários extensos é uma tarefa que exige muitos testes com o software.

1.3- ESTRUTURA

Esta monografia está dividida em cinco capítulos, sendo o primeiro deles esta introdução. O segundo capítulo dedica-se a uma revisão da literatura, indicando os principais trabalhos desenvolvidos recentemente na área de gerenciamento de diálogos, e nas especificações de cada módulo. O terceiro capítulo dedica-se a apresentar os materiais e métodos utilizados para o desenvolvimento do trabalho, mostrando o robô minerva e a arquitetura do sistema que foi implementada.

No quarto capítulo são apresentados os resultados obtidos, tanto individualmente por módulo, quanto o resultado geral e os diálogos obtidos com o robô. No último capítulo constam as conclusões obtidas com o trabalho.

2- REVISÃO DA LITERATURA

2.1- GERENCIADORES DE DIÁLOGOS

Desde o final da década de noventa, a comunidade científica tem obtido avanços no estudo da comunicabilidade humana e na criação de gerenciadores de diálogo para robôs e bots de conversação virtuais.

Entre os anos de 1995 e 2002 foi desenvolvida a AIML (*Artificial Intelligence Markup Language*). A AIML é uma linguagem baseada em XML. Ela foi criada por Richard Wallace em conjunto com a comunidade de software livre Alicebot e constituiu a base para a criação do primeiro Alicebot: A.L.I.C.E. (*Artificial Linguistic Internet Computer Entity*). O projeto da AIML é baseado nos seguintes objetivos (WALLACE, 2005):

1. A AIML deve ser fácil de aprender;
2. A AIML deve codificar o menor conjunto de conceitos necessários para permitir o sistema de conhecimentos para resposta a estímulo do A.L.I.C.E. original;
3. A AIML deve ser compatível com XML;
4. Deve ser fácil escrever programas para processar documentos AIML;
5. Objetos AIML devem ser legíveis por humanos e razoavelmente claros;
6. O projeto de AIML deve ser formal e conciso.
7. Não deve haver dependências entre AIML e outras linguagens.

Para exemplificar o funcionamento da AIML considere o exemplo da Figura 3- Exemplo de categoria em AIML:

```
<category>
<pattern>OI</pattern>
<template><random>
  <li>Oi, tudo bem?</li>
  <li>Oi, como vai você?</li>
  <li>Olá!!</li>
  <li>Oi! Como vai?</li>
  <li>Oi, tudo certo?</li>
</random></template>
</category>
```

Figura 3- Exemplo de categoria em AIML

Neste exemplo nota-se que a AIML é uma *XML-application*. As principais *tags* são: *category*, *pattern* e *template*. Cada categoria possui um conjunto entrada e saída. Os *pattern* são padrões de entrada, por exemplo, quando a entrada for “Oi”, a resposta será proveniente

da categoria cujo *pattern* é “Oi”. O *template* possui a resposta, ou o conjunto de possíveis respostas. No exemplo acima quando o interlocutor disser “Oi”, o programa em AIML retornará aleatoriamente uma dentre as possíveis respostas listadas: “Oi, tudo bem?”, “Olá!!”, ou qualquer uma das outras contidas na lista.

O exemplo mostra apenas algumas das *tags* que constituem a AIML, mas já apresenta a ideia principal: para cada estímulo há um conjunto de respostas. Apesar da simplicidade da linguagem, muitos robôs virtuais disponibilizados na internet possuem relativa assertividade, pois são aprimorados constantemente através de novas categorias que são inseridas por internautas. O *ChatterBox Challenge* é um desafio que acontece anualmente onde internautas do mundo inteiro apresentam seus *chatbots* para avaliação. Aos *chatbots* é feita uma série de perguntas e anotadas as repostas, sendo que aquele que obtém o melhor desempenho, segundo a opinião dos juízes da competição, é o campeão. (CHATTERBOX CHALLENGE, 2012).

Entretanto, mesmo observando o que já foi desenvolvido com a linguagem, os recursos oferecidos por ela para gerenciamento de diálogos ainda é muito simples, já que deixa de apresentar diversas características do diálogo humano, tais como: interpretação do discurso, memória episódica e memória semântica, detecção de emoções e eventos e detecção de ordens ou tarefas enviadas pelo discurso.

A principal aplicação que tem sido observada para a AIML é no desenvolvimento de ambientes de aprendizado virtual. Em (GALVÃO *et al.*, 2004) é criada uma arquitetura para desenvolvimento de um *chatbot* AIML com personalidade. Em (MIKIC *et al.*, 2009) é proposto um sistema educacional inteligente, no qual um *chatbot* denominado CHARLIE realiza a interação entre o estudante e o sistema. Em (ALENCAR E NETTO, 2011) é utilizada a linguagem para criar um ambiente virtual de aprendizado que auxilia alunos de cursos a distância a sanarem suas dúvidas quando os tutores não estão disponíveis.

Nos próximos parágrafos serão apresentadas algumas arquiteturas de gerenciadores de diálogos mais complexos presentes na literatura e suas aplicações. Por fim será apresentada a arquitetura proposta em (ALFENAS E BARRETTO, 2012), que foi utilizada como base para a proposta de arquitetura de gerenciamento de diálogos usada neste trabalho.

Em (O´SHEA *et al.,* 2010) é proposto um agente de conversação (CA - *conversational agent*), voltado para aconselhar estudantes em débito com a universidade.

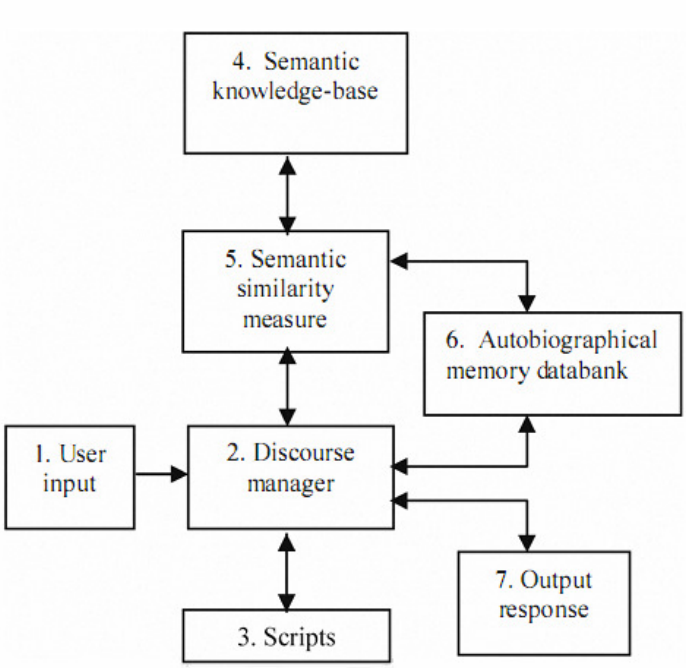

Figura 4 – Arquitetura proposta de (O´SHEA *et al.*, 2010)

Essa arquitetura (Figura 4) caracteriza claramente o Gerenciador de Diálogo (*Discourse manager*). O bloco *Autobiographical memory databank* utiliza a memória episódica, porém restrita à conversação corrente. Já o bloco *Semantic similarity measure* identifica similaridade de conceitos para determinação do contexto da conversação, com base na distância entre conceitos contidos na base de dados de memória semântica (*Semantic knowledge base*). Em (ALFENAS E BARRETTO, 2012) é avaliado que o gerenciador se assemelha ao utilizado com AIML, pois identifica-se uma situação a partir do conceito mais similar encontrado e da estrutura da frase, gerando-se a saída a partir de um *template* de resposta.

Em (KASAP E MAGNENAT-THALMANN, 2010) propõem uma arquitetura (Figura 5) para auxiliar estudantes no aprendizado de redes de computadores. O gerenciador de diálogos está identificado claramente na arquitetura, assim como os módulos de memória. Aqui o gerenciador é baseado em HTN (*Hierarquical Task Network*), uma técnica que escolhe a situação de diálogo a aplicar. Em seguida, máquinas de estados finitos (FSM, *finite state machine*) são utilizadas para a condução do diálogo propriamente dito.

A arquitetura, entretanto não foge à estrutura fixa dos diálogos, por causa do uso das FSM. Um ponto importante dessa arquitetura é a consideração o sentimento do robô em relação ao interlocutor e o histórico de interrelações entre os dois. Isso é representado no diagrama da Figura 5 pelos blocos *Emotion Engine* e *Relationship Calculator*.

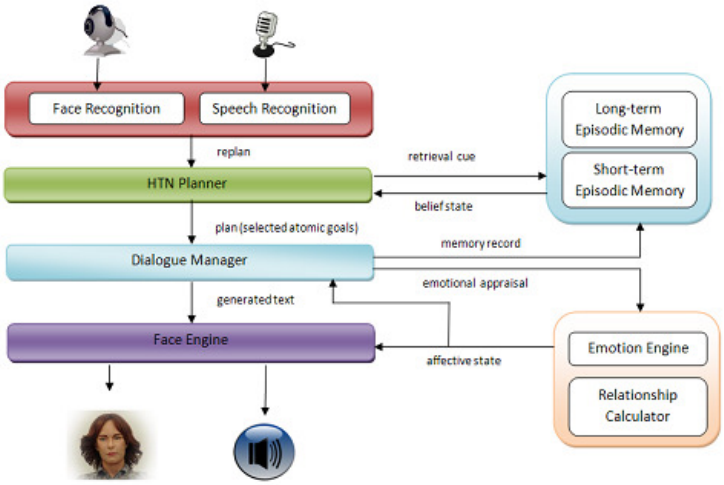

Figura 5 – Arquitetura proposta em (KASAP E MAGNENAT-THALMANN, 2010)

Em (ALFENAS E BARRETTO, 2012) é proposta uma arquitetura de gerenciador de diálogos para um robô sociável. Esta arquitetura, ilustrada na Figura 6, é uma referência para a construção de um sistema de conversação com utilização de adaptatividade no gerenciamento do diálogo, incluindo memórias episódica e semântica.

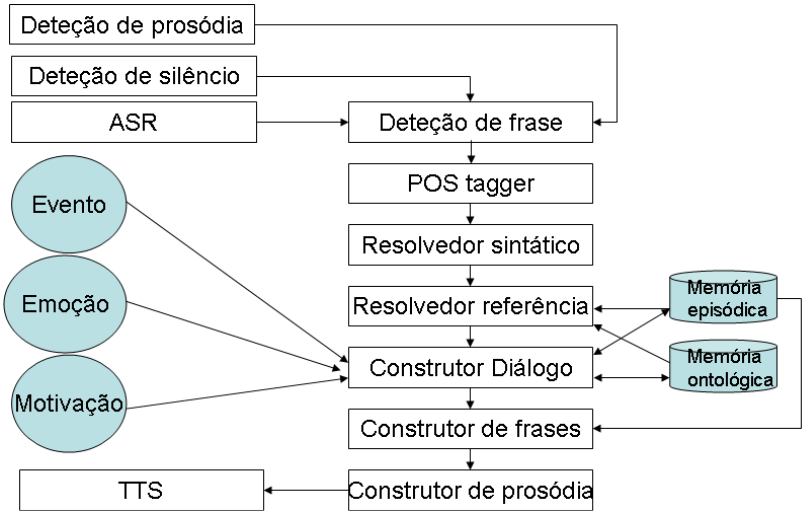

Figura 6 – Arquitetura do Gerenciador de Diálogos proposta em (ALFENAS E BARRETTO, 2012)

Para produzir um comportamente semelhante ao de um ser humano são utilizadas memórias semântica e adaptativa, combinadas a um Gerenciador de Diálogos adaptativo. A adaptatividade do Gerenciador de Diálogos é proveniente da utilização de Máquinas de Markov Adaptativas.

Como pode ser observada na Figura 6, a arquitetura é composta pelos seguintes elementos essenciais:

1. ASR (*automatic speech recognition*): responsável por transformar a fala do interlocutor em texto.

2. Deteção de silêncio: módulo auxiliar, visa auxiliar na identificação de frases completas e na atitude pro-ativa do robô em iniciar uma conversa se não houver intervenção do interlocutor.
3. Deteção de prosódia: determina o tipo de frase (afirmativa, interrogativa, exclamativa) que é pronunciada pelo interlocutor.
4. Deteção de frase: normaliza as saídas dos módulos anteriores, contendo pontuação completa.
5. POS (*part-of-speech*) *tagger*: detecta a classe de cada palavra, sua função gramatical, flexão e forma primitiva.
6. Resolvedor sintático: resolve as relações sintáticas das palavras
7. Resolvedor de referência: resolve as referências contextuais do diálogo, fazendo uso principalmente da memória episódica. Notar que, na arquitetura proposta, a memória episódica assume também as funções de memória de curto prazo.
8. Construtor de Diálogo ou Gerenciador de Diálogo: responsável pela condução da conversação.
9. Construtor de frases: responsável pela transformação da saída do Gerenciador de Diálogo em uma frase completa.
10. Construtor de prosódia: responsável por adicionar elementos prosódicos à saída.
11. TTS (text-to-speech): responsável pela geração da fala propriamente dita.
12. Memória episódica: responsável pela memória episódica, como conceituada anteriormente.
13. Memória ontológica: trata-se da memória semântica, mas assim denominada porque, na proposta, este módulo será construído a partir de ontologias.

A arquitetura prevê ainda os módulos de:

- Emoção – responsável pela detecção da emoção demonstrada pelo interlocutor. Este módulo já conta com implementações;
- Evento – responsável pela detecção de eventos, como a aproximação ou chegada de um interlocutor ou mesmo extra-conversação, como um ruído muito forte;
- Motivação – responsável pela determinação da tarefa a ser realizada pelo robô. Uma implementação, está sendo realizada.

A Figura 7 apresenta um exemplo de possível diálogo com o robô utilizando a arquitetura proposta, incluindo os conceitos de adaptatividade.

R	1. Decisão MMA: Selecionar CRK (conceito-relação-keyword) a partir do conceito corrente (inicialmente, igual a Pessoa) 2. Selecionado: Pessoa-LIKE-Cachorro 3. Perguntar: Você gosta de cachorro?
J	Sim! (adicionar a WoO::J:: J-LIKE(v=1.0,w=1.0)-Cachorro
R	1. Adicionar a WoO::J:: J-LIKE(v=1.0,w=1.0)-Cachorro 2. Decisão MMA: Selecionar RRR (relação-relação-relação) a partir da relação corrente (LIKE) 3. Selecionado LIKE-OWN 4. Perguntar: Você possui um cachorro?
J	Sim!
R	1. Adicionar a WoE::J::J-OWN(v=1.0,w=1.0)-Cachorro 2. Erro: não pode criar identificador para Cachorro porque não conhece o nome do cachorro. 3. Perguntar: Qual o nome do cachorro?
J	Gaia.
R	1. Adicionar a WoE::J::J-OWN(v=1.0,w=1.0)-Cachorro 2. Decisão MMA:Selecionar CRK a partir do conceito corrente 3. Selecionado Pessoa-Like-Gato. 4. Perguntar: Você gosta de gato?

Figura 7- Exemplo de um possível diálogo utilizando-se a arquitetura proposta em (ALFENAS E BARRETTO, 2012)

2.2- GERAÇÃO DE RESPOSTAS COM AIML

Este item dedica-se a explicar o funcionamento da AIML, mostrando os principais recursos e configurações que permitem o desenvolvimento de *chatbots*. Todas as informações contidas nesta seção são apresentadas em (WALLACE, 2005).

A AIML é uma linguagem XML desenvolvida para gerar diálogo entre o ser humano e o computador. Ela permite associar respostas para cada entrada, gerando assim uma conversação. Um ponto positivo da AIML é a simplicidade de programação e de leitura do código. O ponto negativo é que é necessário prever milhares de possíveis entradas do sistema e acrescentar uma resposta para cada uma.

Conforme explicado na seção 2.1-, a AIML é composta de categorias (*tag category*), onde cada categoria possui um padrão (*tag pattern*) e um *template* de resposta (*tag tamplate*). Além dessas, há ainda outras *tags* e recursos que permitem a criação de diálogos mais naturais. A seguir serão listados alguns recursos utilizados:

1- Frases incompletas

Uma situação muito comum é que a frase dita pelo usuário não seja exatamente igual àquela contida em algum *pattern*. Para evitar esse tipo de situação pode ser

colocado o símbolo * dentro do *pattern* para substituir um eventual trecho de texto. Para facilitar a compreensão é dado um exemplo na Figura 8:

```
<category>
<pattern>* SONO</pattern>
<template>
Eu não durmo muito, no máximo entro no modo de espera.
<template>
</category>
```

Figura 8- Exemplo de uso do símbolo * nos patterns AIML

Neste caso, qualquer trecho de frase que termine com a palavra sono terá como resposta o que se encontra no *template*. Por exemplo, para as frases "Você está com sono" ou "Você sente sono" será dada a mesma resposta.

2- *Tag* <srai>

A *tag* <srai> é utilizada quando para diversas entradas diferentes há uma mesma saída. A principal utilização é no caso de sinônimos ou frases que possuem a mesma resposta.

```
<category>
    <pattern>COMO VOCÊ ESTÁ</pattern>
    <template>Eu estou ótima. E você, como vaí?
    </template>
</category>

<category>
    <pattern>COMO VAI VOCÊ</pattern>
    <template><srai>COMO VOCÊ ESTÁ</srai>
    </template>
</category>

<category>
    <pattern>OI TUDO BEM</pattern>
    <template><srai>COMO VOCÊ ESTÁ</srai>
    </template>
</category>
```

Figura 9- Exemplo de aplicação da *tag* <srai>

Neste caso, apesar de todos os padrões serem diferentes, pode ser dada a mesma resposta para todos, no caso a resposta de "como você está?".

3- *Tag* <that>

Esta *tag* é muito útil para criar uma contextualização, ou seja, para dar uma resposta particular a um padrão quando estiver num contexto específico. Segue o exemplo:

```
<category>
<pattern>SIM</pattern>
<that>VOCE GOSTA DE FILMES</that>
<template>Qual é o seu filme favorite?</template>
</category>
```

Figura 10- Exemplo de aplicação da *tag* <that>

Neste caso, quando a pessoa diz sim, o gerenciador avalia se a última coisa que o *chatbot* disse foi "Você gosta de filmes?", em caso afirmativo dá a resposta "Qual é o seu filme favorito?".

4- *Tags* <set> e <get>

As *tags* set e get são utilizadas para armazenar informações sobre a pessoa com quem o robô está falando. Essas informações podem ser nome, idade, ou qualquer outra informação que se desejar.

```
<category>
<pattern>MEU NOME É *</pattern>
<template>
Certo, vou te chamar de <set name="name"><star/></set>.
</template>
</category>

<category>
<pattern>VOCE GOSTA DE MIM</pattern>
<template>
Eu gosto muito de você <get name="name"/>.
</template>
</category>
```

Figura 11- Exemplo de aplicação das *tags* <set> e <get>

No exemplo da Figura 11, se a pessoa diz "Meu nome é João", a resposta do robô é "Certo, vou te chamar de João.". A *tag* <star/> contém o texto correspondente ao símbolo '*', do *pattern*, no caso, "João". A *tag* <set>, neste caso, está armazenando o texto de <star/> e classificando-o como nome. Isso permite que o nome seja usado novamente depois, como na segunda categoria, usando a *tag* <get>. A segunda categoria do exemplo ilustra essa aplicação, no qual a *tag* <get> é utilizada para capturar o nome e colocar na frase: "Eu gosto muito de você João".

Além dessas, há ainda mais *tags* disponíveis na AIML, porém não se faz necessário expor todas nesta monografia.

O funcionamento do *chatbot* em AIML depende também de configurações através dos arquivos properties.xml, context.xml e substitutions.xml.

O arquivo properties.xml indica o caminho para os principais arquivos do AIML. Neste caso todos os arquivos AIML contendo as categorias estão em "C:\AIML_Ready\ AIML\BotFiles\". Todos os arquivos *.aiml nesta pasta são considerados.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="context">C:\AIML_Ready\AIML\context.xml</entry>
  <entry key="aiml">C:\AIML_Ready\AIML\minerva.xml</entry>
  <entry key="splitters">C:\AIML_Ready\AIML\splitters.xml</entry>
  <entry key="substitutions">C:\AIML_Ready\AIML\substitutions.xml</entry>
  <entry key="categories">C:\AIML_Ready\AIML\BotFiles\</entry>
</properties>
```

Figura 12- Exemplo de arquivo properties.xml

O arquivo substitutions.xml é responsável por substituições do texto que o AIML recebe para o texto que será comparado com os padrões. É útil para padronizar palavras que podem ser utilizadas de forma abreviada na linguagem coloquial.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--Substitutions are grouped according to several AIML interpreter
functions.-->
<substitutions>
  <input>
    <correction><!--sentence correction-->
      <substitute find="tá" replace="está"/>
      <substitute find="to" replace="estou"/>
    </correction>
</substitutions>
```

Figura 13- Exemplo de arquivo substitutions.xml

No exemplo acima as palavras "tá" e "to" são substituídas por "está" e "estou".

No arquivo context.xml é possível armazenar informações sobre o robô e informações sobre a pessoa com quem o robô fala. As informações do robô são estáticas, ou seja, não podem ser alteradas durante a conversação. Essas informações são armazenadas através da *tag* <bot>. Já as informações do falante podem ser alteradas durante o diálogo através da *tag* <set>.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<context>
  <!-- The id is a unique string that identifies this context. -->
  <bot name="id" value="Minerva"/>
  <bot name="name" value="MINERVA"/>
  <bot name="master" value="Professor Barretto"/>
  <bot name="favoritebook" value="O Senhor dos Anéis"/>

  <!-- Bot predicates are set at load time, and cannot be changed at runtime. -->
  <bot name="output" value="output.txt"/>
  <!-- <bot name="randomSeed" value="1"/> values are more random if the seed isn't
specified. -->
  <bot name="series" value="00 (Alpha)"/>
  <bot name="version" value="007"/>

  <!-- Default values for predicates, can be changed later at runtime. -->
  <set name="dateFormat" value="yyyy-MM-dd HH:mm:ss"/>
  <set name="name" value="pessoa desconhecida"/>
  <set name="idade" value="23 anos"/>
</context>
```

Figura 14- Exemplo de arquivo context.xml

No exemplo da Figura 14- Exemplo de arquivo context.xml é definido o nome do robô como Minerva, o nome do mestre como Professor Barreto e o livro favorito como O Senhor dos Anéis. O nome da pessoa com quem o robô conversa é definido como pessoa desconhecida e a idade como vinte e três anos.

2.3- RECONHECIMENTO AUTOMÁTICO DE FALA

A revisão da literatura no tema de reconhecimento automático de fala será sucinta nesta monografia, uma vez que não faz parte do escopo do trabalho o desenvolvimento de um novo sistema de reconhecimento de fala. No trabalho será utilizada uma solução comercial, o Loquendo ASR, cujo funcionamento também será explicado nesta seção.

O reconhecimento automático de fala (ASR – *Automatic Speech Recognition*) tem sido um tema investigado no processo de fala ao longo das últimas décadas. De forma geral, pode ser definido como o reconhecimento (não entendimento) de palavras de um dado dicionário proferidas por um falante, confiando apenas na informação contida no sinal de fala proferido e no conhecimento prévio no domínio do problema (TRENTIN, 2000).

Na década de cinquenta, quando o problema começou a ser estudado, pensou-se que o problema seria facilmente resolvido com novas tecnologias computacionais, entretanto décadas se passaram e o tema despontou como um problema difícil de resolver. Até hoje muitas questões difíceis estão em aberto, apesar do esforço dos pesquisadores para resolvê-las. As dificuldades estão relacionadas a aumentar o tamanho dos dicionários, ao compromisso entre reconhecimento de discurso contínuo e reconhecimento de palavras isoladas, às características vocais do falante em reconhecedores independentes do falante (SI

– *speaker independent*), à capacidade de reconhecer somente o discurso proferido pelo falante que treinou o reconhecedor, ao fenômeno do discurso espontâneo (um's, ah's, falsos começos, palavras fora do vocabulário, etc.), à robustês às condições do ambiente (ruídos e distorções no canal), entre outros problemas (TRENTIN, 2000).

O problema do ASR pode ser formulado como um problema de classificação estatística, de acordo com o reconhecimento clássico de padrões. Uma vez que as classes foram definidas como sequências W de palavras permitidas de um dicionário "fechado", uma representação paramétrica do sinal do discurso foi escolhida (e.g. uma sequência de vetores de características acústicas X), e um critério de *Maximum a Posteriori* (MAP) foi adotado, o problema de classificação pode ser estabelecido como a procura pela sequência de palavras W* que maximiza a quantidade $Pr(\mathrm{W}|\mathrm{X})$. Isto pode ser expresso pelo teorema de Bayes da seguinte forma (TRENTIN, 2000):

$$Pr(\mathrm{W}|\mathrm{X}) = \frac{Pr(\mathrm{X}|\mathrm{W})\,Pr(\mathrm{W})}{Pr\,(\mathrm{X})} \tag{1}$$

Dada uma sequência de obervações X, os esforços para maximizar $Pr(\mathrm{W}|\mathrm{X})$ podem concentrar-se em procurar pela classe W* que maximiza o numerador do termo da direita na Equação 1, i.e. $Pr(\mathrm{X}|\mathrm{W})\,Pr(\mathrm{W})$. A quantidade $Pr\,(\mathrm{X})$, usualmente denominada como *language model* (LM), depende das restrições de alto nível e do conhecimento liguístico sobre as palavras permitidas para a específica tarefa. A quantidade $Pr(\mathrm{X}|\mathrm{W})$ é conhecida como *acoustic model*. Ela descreve a estatística de sequências de observações acústicas parametrizadas no espaço de características dadas as palavras proferidas correspondentes (e.g. certos fonemas) (TRENTIN, 2000).

O Modelo de Markov Escondido (HMM – *Hidden Markov Model*) é um modelo paramétrico em nível acústico. É uma abordagem efetiva para o problema de modelamento acústico em ASR, permitindo bom desempenho de reconhecimento em muitas circunstâncias. Uma Rede Neural Artificial (ANN – *Artificial Neural Network*) é um grupo interconectado de neurônios artificiais que usa um modelo computacional de processamento baseado numa abordagem conexionista. As ANN's são amplamente utilizadas em muitos campos da engenharia, em particular no reconhecimento de padrões, na classificação e na predição (GEMELLO *et al.*, 2009).

Neste trabalho, será utilizado o software comercial Loquendo ASR 7.10. A tecnologia utilizada é um híbrido de HMM e ANN (vide Figura 15). O sistema combina a estrutura de modelagem sequencial do HMM, capaz de lidar com padrões temporais e o poder de reconhecimento de padrões do ANN. Como no HMM padrão, o híbrido HMM/ANN aplicado

ao ASR usa o processo de Markov para o modelo temporal do sinal de fala. A estrutura conexionista das ANN é usada para modelar o vetor local de características condicionada no processo de Markov. O processo híbrido é baseado na teoria de que as ANN podem estimular probabilidades de classe para padrões de entrada. Essa probabilidade pode ser usada, depois de algumas movimentações, como probabilidades locais no HMM (GEMELLO *et al.*, 2009).

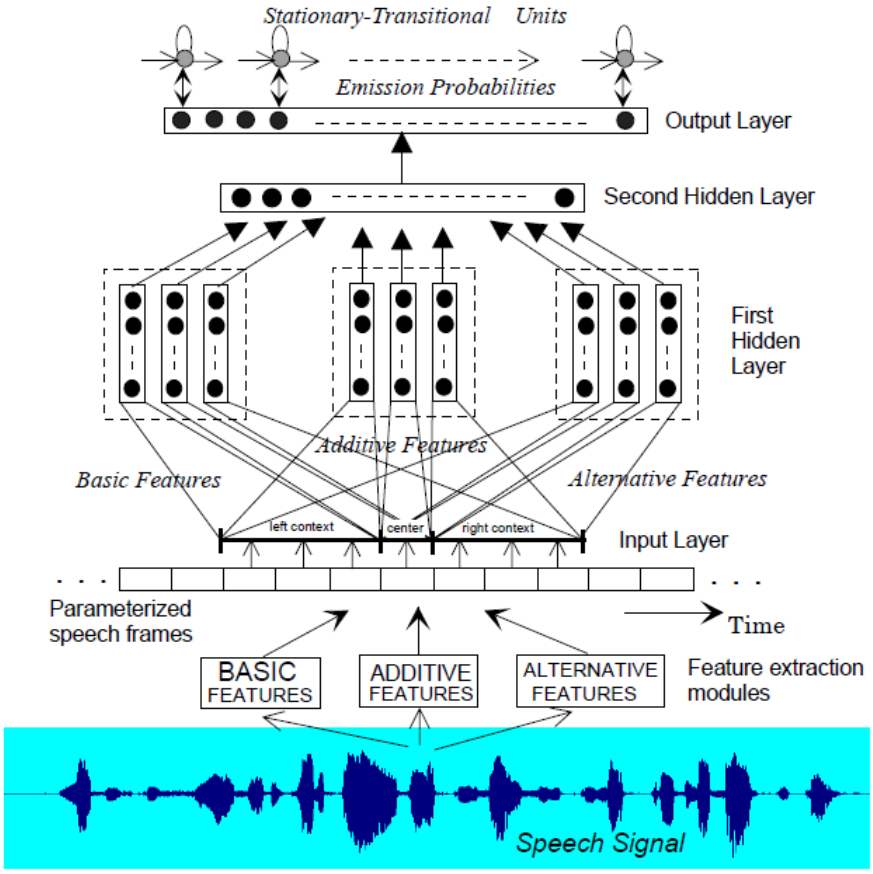

Figura 15 - Arquitetura geral do medlo Loquendo Hybrid HMM/ANN (GEMELLO *et al.* , 2009)

Antes de ser realizada a classificação, o sinal sonoro passa por uma série de processamentos de sinais. O processamento deve remover todas as informações irrelevantes como ruídos de fundo e características do dispositivo de gravação e codificar a informação relevante que sobra numa série de atributos que são usadas como entradas no classificador. Atributos, ou *features*, podem ser definidos como uma unidade mínima que distingue classes maximamente próximas (KUMAR *et al.*, 2010).

No Loquendo ASR, esses atributos usados na caracterização do sinal sonoros são de três tipos:

1. Basic Features: são usados dois coeficientes independentes, o MFCC (*Mel-frequency Cepstral Coefficient*) e o RPLP (*Revised Perceptual Linear Prediction Coefficient*)

2. Additive Features: esse atributos não são independentes, mas devem ser utilizados como acréscimo aos *basic features*, dentre eles: tom, periodicidade, centros de gravidade, *formants*, e a razão sinal-ruído.
3. Alternative Features: são atributos suficientemente informativos que podem ser usados em substituição aos *basic features*. Como exemplo existem: *Wavelet derived parameters*, *ear-model features*.

Com o Loquendo ASR é possível desenvolver aplicativos de reconhecimento de fala através de um conjunto de API's (*Application Programming Interface*) de reconhecimento. Ele possibilita criar aplicativos que utilizam tecnologias complexas de reconhecimento e obter resultados elaborados de pós-processamento utilizando um conjunto reduzido de API's e de parâmetros (LOQUENDO, 2011c).

O software é construído sobre um *kernel* que usa tecnologia de vocabulários flexíveis. Essa tecnologia permite:

- Uso do mesmo reconhecedor de fala em sistemas com diferentes vocabulários;
- Modificações rápidas e frequentes e customizações no conjunto de palavras a serem reconhecidas;
- Qualquer palavra pode ser definida como uma sequência de unidades acústico-fonéticas, cada uma representando um fenômeno elementar do discurso;
- Distinção de palavras acusticamente muito semelhantes.

A entidade que contém toda a informação necessária para que o reconhecedor realize a ação de reconhecimento é denominada de Objeto de Reconhecimento (RO – *Recognition Object*). Um RO é sempre o resultado do processo de compilação de um conhecimento fonte, que pode ser uma gramática ou um conjunto de palavras na forma grafêmica. O máximo número de palavras que podem ser incluídas num RO é de 100.000 (LOQUENDO, 2011c).

Em muitos casos o resultado do reconhecimento é insatisfatório e deve ser rejeitado. O Loquendo ASR possui um mecanismo de rejeição baseado em confiança. Os resultados de reconhecimento incluem o valor de confiança (*confidence value*) e a resposta de rejeição e é possível saber qual técnica de rejeição sugeriu essa rejeição. O valor de confiança é um valor em ponto flutuante que representa a confiabilidade do reconhecimento. O valor está entre 0.0 e 1.0, de tal modo que 0.0 significa que o resultado não é confiável, e 1.0 indica que o sistema está muito confiante da assertividade do resultado.

O desempenho dos aplicativos pode ser melhorado através da ferramenta de aprendizado fonético. Essa ferramenta busca melhorar a efetividade de objetos de reconhecimento gramaticais através de duas questões principais (LOQUENDO, 2011c):

- Descoberta automática de variantes de pronúncia para palavras contidas na gramática;
- Agrupamento de formulações linguísticas que não são frequentemente previstas.

Para criar um aplicativo utilizando as API's do Loquendo ASR é necessário seguir a seguinte sequência de passos:

1. Configurar o ASR: selecionando a língua, as pastas de destino para os objetos de reconhecimento e outras configurações;
2. Iniciar uma seção com essas configurações e em seguida uma instância para esta seção;
3. Registrar na instância a fonte de áudio, no caso o microfone, e o *Event Listener*, que trata os eventos ocorridos;
4. Compilar a gramática;
5. Selecionar os parâmetros da instância, como modo de reconhecimento, *timeouts*, velocidade, entre outros;
6. Realizar o reconhecimento;
7. Adquirir os resultados.

Como resultados o Loquendo ASR apresenta as hipóteses de reconhecimento, ordenadas da mais provável para a menos provável. Cada hipótese contém as palavras e o valor de confiança de cada palavra individualmente. O programa também apresenta a variável *Rejection*, que sugere a rejeição ou não da hipótese escolhida, baseado no seu valor de confiança. Por fim o SNR, *Signal to noise ratio*, que é uma medida do ruído do sinal capturado.

O Loquendo ASR só é capaz de detectar palavras contidas num vocabulário. Ou seja, quando um discurso é detectado, as características do discurso são comparadas com aquelas esperadas para as palavras do vocabulário, quanto maior a semelhança, maior o nível de confiança de que o discurso corresponda a uma determinada palavra ou frase. Além do vocabulário, podem ser determinadas classes de palavras e regras gramaticais para criação de

frase. O vocabulário e as regras gramaticais são incluídas num arquivo chamado gramática cuja extensão é *.gram. Um exemplo de gramática pode ser visto na Figura 16:

```
#ABNF 1.0 UTF-8;

meta "loq-remark" is "frase";
meta "loq-globmodel" is "DEFAULT";

language  pt-br;
mode  voice;
root  $frase;
tag-format <semantics/1.0>;

$frase = (  $artigo  $substantivo  $verbo  $adverbio |
              $adverbio  $verbo  $artigo  $substantivo
          );

$artigo = ( a | o );
$substantivo = ( amigo | mãe | filha );
$verbo = ( chegou | dormiu );
$adverbio = ( cedo | tarde );
```

Figura 16- Exemplo de gramática do Loquendo ASR

Neste exemplo são estabelecidas quatro classes de palavras: artigo, substantivo, verbo e advérbio. As possibilidades de combinação são determinadas pela classe frase e nesse caso, podem ser formadas frases como: “a mãe dormiu cedo” ou “tarde chegou o amigo”.

A tarefa de reconhecimento se torna mais complicada à medida que o tamanho das gramáticas e dos vocabulários a serem reconhecidos são muito grandes. Em particular quando é permitido ao falante dizer frases completas, e especialmente quando pronunciadas de uma forma espontânea. Cada língua tem milhões de possíveis combinações de palavras e frases, e as gramáticas devem ser robustas o suficiente para cobrir isso (BAGGIA *et al.*, 2009).

Nesse caso, é aconselhável o uso de modelos de linguagem ou gramáticas para restringir as combinações admissíveis de palavras. Essas limitações são aplicadas em tempo real durante o reconhecimento para restringir a área de pesquisa; o uso seguinte de reticulados de hipóteses de palavras produzido pelo ASR não produz os mesmos resultados. Até mesmo os humanos utilizam o procedimento de palavras precedentes e sucessivas para determinar qual palavra acabou de ser dita (BAGGIA *et al.*, 2009).

O Loquendo ASR disponibiliza ainda algumas configurações que permitem o ajuste da maneira como a detecção é realizada, conforme a necessidade do aplicativo. Uma configuração que pode ser definida é o modo de reconhecimento, que pode ser um entre cinco tipos: normal, *silent*, *semisilent*, *silent_restart* e *semisilent_restart*. Esses modos determinam o comportamento do detector de acordo com os seguintes casos: palavras dentro do

vocabulário (INV), palavras fora do vocabulário (OOV) e excesso de ruído (BGN). O comportamento é determinado pela tabela seguinte (LOQUENDO, 2011c):

Tabela 1 – Módulos de Reconhecimento (LOQUENDO, 2011c)

	INV	**OOV**	**BGN**
NOMAL	<Results>	<Rejection>	<NoResult>
SEMISILENT	<Results>	<Rejection>	<Restart>
SILENT	<Results>	<Restart>	<Restart>
SEMISILENT_RESTART	<Results> + <Restart>	<Rejection> + <Restart>	<Restart>
SILENT_RESTART	<Results> + <Restart>	<Rejection> + <Restart>	<Restart>

Existe ainda uma série de parâmetros configuráveis para o reconhecedor, dentre os quais os listados a seguir (LOQUENDO, 2011b):

- Speed: velocidade do reconhecimento, quanto mais rápido maior a incerteza no reconhecimento;
- Speech Incomplete Timeout: é o comprimento do silêncio necessário após o discurso para que o reconhecedor finalize o resultado;
- Speech Complete Timeout: o mesmo que o incomplete timeout, porém é utilizado quando há correspondência completa do discurso proferido com a gramática e não serão mais proferidas novas palavras.
- Speech Timeout: É o máximo tempo de discurso, em ms.
- Silence Timeout: É o máximo tempo que o reconhecedor espera até que algum discurso seja identificado, em ms.
- Audio Timeout: É o máximo de vezes consecutivas que o reconhecedor requisita novas amostras de áudio sem obter sucesso, quanto está usando fonte de áudio.
- Audio Stop Timeout: É o máximo de vezes consecutivas que o reconhecedor requisita uma fonte de áudio se esta está parada.
- Resource Timeout: É o tempo para acessar recursos.

2.4- GERAÇÃO DE PROSÓDIA

Uma das definições para prosódia apresentada em (MATEUS, 2004) é retirada do Dicionário de Termos Linguísticos: prosódia é o “estudo da natureza e funcionamento das

variações de tom, intensidade e duração na cadeia da fala". Onde tom, intensidade e duração são propriedades inerentes ao som.

A variação destas e de outras propriedades do som são responsáveis pela prosódia e consequentemente pela natureza da fala humana que não é monotônica e constante, mas varia a cada elemento fonético dependente de diferentes fatores como saúde, região, idioma, emoções entre outros.

Introduzir prosódia num texto falado por um robô pode aumentar a verossimilhança da voz com a natural humana (REIS *et al*,2010), o que é um grande desafio atualmente. Para inserir controles de prosódia em um texto, foram criadas especificações para uma Markup Language no formato XML (BURNETT, 2002) pela W3C, visando melhorar as saídas de voz principalmente em aplicações na internet. Com o objetivo de padronizar os controles e após validações, foi feita a primeira versão oficial das especificações (BURNETT, 2004).

Como dito anteriormente, essas especificações seguem um formato da XML, onde o arquivo texto contém *tags* de controle, como é possível observar no exemplo da Figura 17 a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<speak version="1.0"
     xmlns="http://www.w3.org/2001/10/synthesis"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
     http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
     xml:lang="pt-BR">
  <p>
    <voice name="Fernanda">

        <prosody pitch="high">Oi, tudo bem?</prosody>

    </voice>
  </p>
</speak>
```

Figura 17- Arquivo SSML para inclusão de prosódia à fala do robô

Na figura 17 é possível notar a estrutura da SSML e algumas das *tags* mínimas necessárias, bem como algumas opções de controle, conforme explicado a seguir:

- *Tags* mínimas necessárias:

```
<?xml version="1.0" encoding="UTF-8"?>
<speak version="1.0"
     xmlns="http://www.w3.org/2001/10/synthesis"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
     http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
     xml:lang="pt-BR">
```

  - Estas *tags* iniciais são necessárias para identificar o formato e torná-lo reconhecível pelo programa que interpreta o texto SSML.

  - Outra *tag* necessária é a que define qual voz será utilizada:

```
<voice name="Fernanda">
```

- *Tags* de controle:
  - Dentre as *tags* de controle, a que controla a prosódia é <prosody (propriedade)=”atributo”>, onde propriedades possíveis são: pitch, contour, range, rate, duration e volume. E atributos possíveis são por exemplo: "x-low", "low", "medium", "high", "x-high", ou "default".

2.5- GERAÇÃO DE FALA A PARTIR DE TEXTO

A revisão da literatura no tema de geração de fala a partir de texto será sucinta nesta monografia, uma vez que não faz parte do escopo do trabalho o desenvolvimento de um novo sistema de conversão de texto em fala. Neste trabalho será utilizada uma solução comercial, o Loquendo TTS, cujo funcionamento também será explicado nesta seção.

Hoje existem diversos sistemas que conseguem reproduzir a voz humana, porém não naturalmente com todos os elementos de prosódia e fonética. Hoje estes sistemas visam principalmente duas características dessa conversão: inteligibilidade e naturalidade. A primeira refere-se à compreensão efetiva do texto pelo receptor, enquanto que a segunda está relacionada à naturalidade da voz, fazendo-a se parecer o máximo possível com a voz humana.

Um sistema de conversão de texto em fala pode ser dividido da seguinte maneira, como indicado na Figura 18.

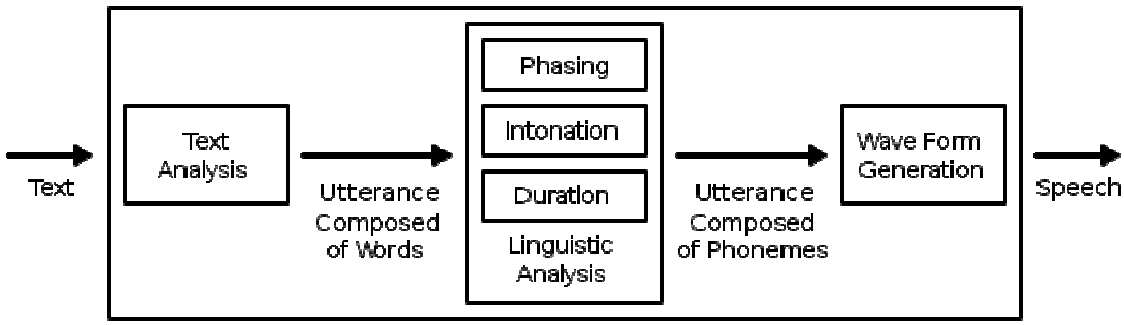

Figura 18 - Divisão de um sistema TTS

O sistema é composto por um módulo de análise do texto, que é responsável por dividir o texto em elementos menores e manejáveis, para permitir e facilitar a análise linguística.

Na análise linguística, para um sistema de TTS que faz a síntese através da concatenação de fonemas, o software faz a conversão dos elementos do texto em fonemas e marcadores de prosódia, intensidade, duração, ênfase, etc.

Depois de realizada a conversão do texto em fonemas, o programa gera as ondas sonoras que representam os sons, convertendo desta forma o texto escrito em fala, e tentando reproduzir o mais próximo da voz humana.

Existem hoje diferentes formas de converter-se o texto escrito em fonemas para reprodução do som. O sistema em questão utilizado para a realização deste trabalho, o TTS da Loquendo® possui uma base gravada de fonemas, ou seja, não existem palavras inteiras armazenadas, o que tornaria necessário o uso de muita memória. O sistema divide as palavras em fonemas individuais, difones, sílabas entre outros elementos individuais, permitindo assim uma maior versatilidade do sistema e aumentando a naturalidade. Esta técnica é denominada de síntese por concatenação.

Atualmente existem também outras formas de síntese, dentre as quais estão (RASHAD *et all*, 2010):

- Síntese de formação: usa um modelo acústico e sínteses aditivas, não utilizando gravações humanas para gerar a saída;
- Síntese articulatória: visa reproduzir o som baseado em modelos do trato vocal humano e no processo de articulação, utilizando modelos computacionais para controlar língua, lábios, palato, glote, etc. e também às pregas vocálicas. Estes modelos são modificados utilizando-se regras com a finalidade de representar os fonemas.

- Síntese baseada em modelos de Markov: é um método baseado em modelos ocultos de Markov, também conhecido como síntese paramétrica estatística. Nesse sistema, o espectro de frequência do trato vocal a fonte de voz, frequência fundamental, e a prosódia do discurso são modelados simultaneamente pelos modelos de Markov. Formas de onda do discurso são geradas pelos modelos através do critério de semelhança.

Estas formas de síntese não serão abordadas neste trabalho por não serem utilizadas no software que será usado para desenvolver o programa responsável pela fala.

2.6- SINCRONIA LABIAL

A sincronização dos lábios é um tema há muito explorado, uma das primeiras tentativas de sincronizar o som com os lábios pode ser obervada na animação: *"My old Kentucky Home"* (1926), uma das primeiras com som (MALTIN, 1980), onde o cachorro do desenho pronuncia a seguinte frase: *"Follow the ball, and join in, everybody"* (Sigam a bola e juntem-se, todos; tradução livre). Após este, outros começaram a seguir o mesmo caminho da utilização e principalmente da sincronização do som, o tema começou a aparecer cada vez mais e em outras frentes como filmes, em dublagens, animações, vídeo games na voz dos personagens cada vez mais tecnológicos e mais verossímeis no comportamento, televisão e até na música, em clipes e show. Com a evolução da tecnologia, da computação gráfica e dos computadores, essa sincronização se mostra cada vez mais verossímil.

Entretanto no campo físico, da sincronização labial de robôs este tema é recente, pois envolve muitas variáveis para simular a musculatura dos lábios e controlar os movimentos, tornando mais fidedigno a realidade, e permitindo sincronizar os movimentos com a fala. Um exemplo é o robô "Kismet" (Figura 19) que dentre outras funções do rosto do robô, incluía movimentação dos lábios para representar emoções e tentar movimentá-los de forma sincronizada com a fala (BREAZEAL, 2002).

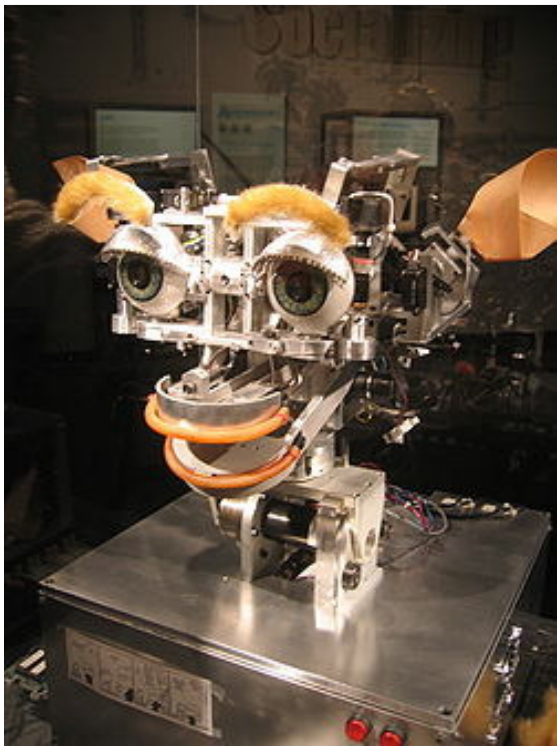

Figura 19 - Robô “Kismet”, projeto de Breazeal, MIT (BENEDICT, 2005).

Há posições para alguns fonemas padrões que pré determinam o tamanho da abertura da boca, podemos observar alguns exemplos de fonemas com suas correspondentes aberturas de boca na Figura 20.

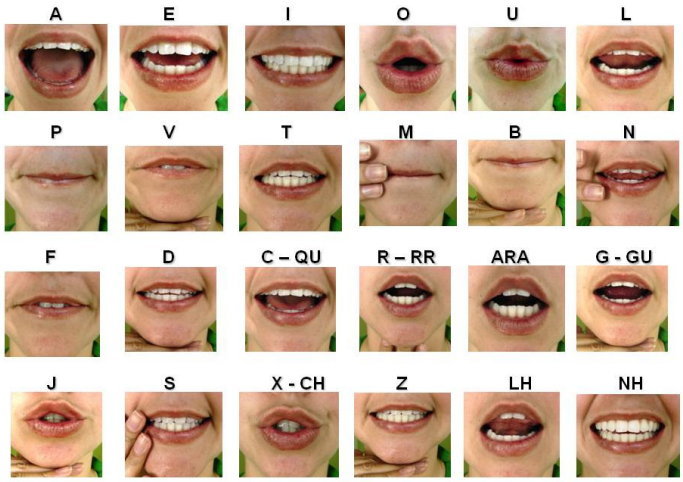

Figura 20 - Formas da boca (posições dos lábios) para alguns fonemas (Método Fonovisuoarticulatório: Bocas – Camilo, Helena, Psicologia 09)

3- MATERIAIS E MÉTODOS

Este capítulo tem como objetivo descrever a arquitetura de geração de conversação desenvolvida, bem como mostrar todos os componentes do sistema. Num primeiro momento será dada uma visão geral do sistema para, em seguida, apresentar o robô e as arquiteturas de hardware e software utilizadas.

3.1- VISÃO GERAL

A arquitetura do gerenciador de diálogos escolhida é uma simplificação da arquitetura proposta em (ALFENAS E BARRETTO, 2012) (Figura 6). Da sequência de módulos apresentada são preservados os seguintes: 1 – ASR, 10 – Construtor de prosódia e 11 – TTS. Todos os módulos intermediários são substituídos por um chatbot AIML, que simplifica a geração do diálogo, uma vez que não inclui memória, interpretação ou detecção de emoções. Cada módulo é desenvolvido de forma independente dos módulos adjacentes e, portanto, o funcionamento do sistema como um todo depende da integração entre essas partes. A Figura 21 apresenta uma breve descrição de cada módulo e de como é o fluxo de informações dentro do sistema.

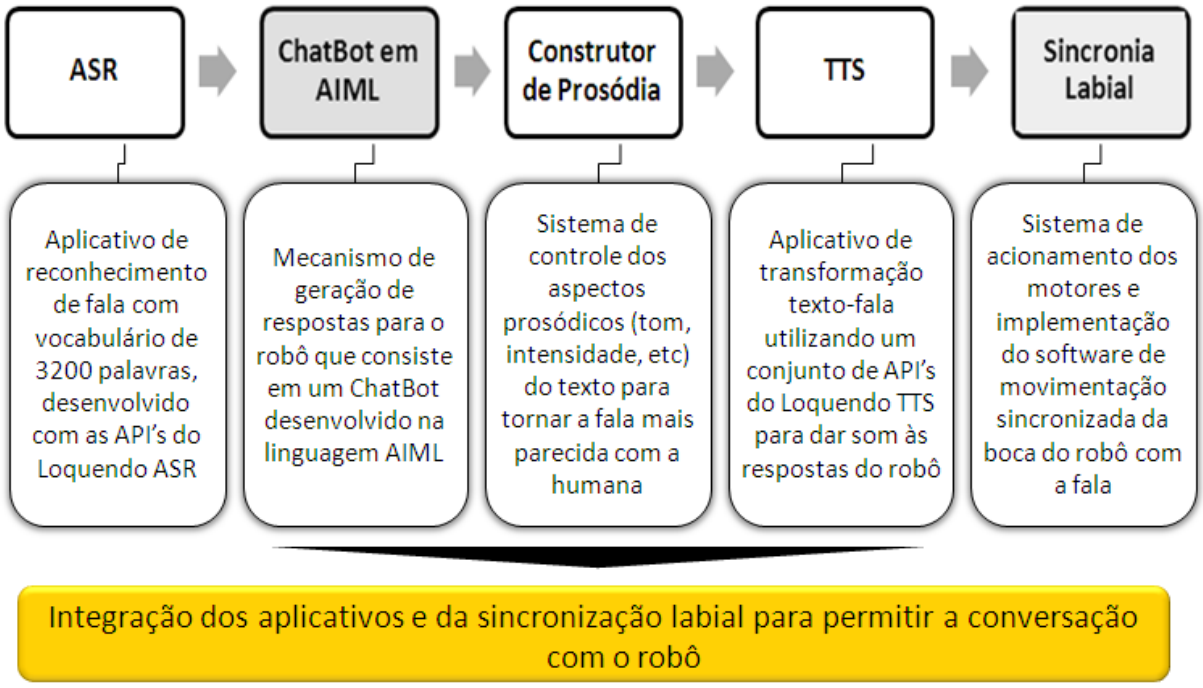

Figura 21- Módulos do Gerenciador de Diálogos

São cinco os módulos do gerenciador:

1. ASR – Reconhecimento automático de fala

- ENTRADA: Áudio obtido da fala de um ser humano
- SAÍDA: Texto correspondente ao discurso proferido
- DESCRIÇÃO: O aplicativo, desenvolvido utilizando as API's do Loquendo ASR tem como objetivo reconhecer o discurso falado em um sinal de áudio e transcrevê-lo na forma de texto da forma mais fiel possível. A maior dificuldade é a identificação de frases quando o vocabulário disponibilizado é muito extenso.

2. *ChatBot* em AIML – Geração da resposta do robô
    - ENTRADA: Texto correspondente à fala do ser humano
    - SAÍDA: Texto correspondente à resposta do robô
    - DESCRIÇÃO: Um conjunto de arquivos em AIML que associam respostas para frases pré-definidas. A linguagem oferece alguns artifícios que podem ser utilizados para facilitar a obtenção de diálogos coerentes.
3. Construtor de Prosódia
    - ENTRADA: Texto de resposta do AIML
    - SAÍDA: Texto com *tags* de no formato de XML (SSML) que orientam a inserção de aspectos prosódicos no discurso de resposta
    - DESCRIÇÃO: Para que o módulo seguinte (TTS) possa transmitir mais naturalidade ao discurso proferido pelo robô, são inseridos no texto *tags* modificam a prosódia na pronúncia das frases ou palavras.
4. TTS – Transformação Texto-Fala
    - ENTRADA: Texto com ou sem *tags* de prosódia
    - SAÍDA: Voz feminina pronunciando o discurso correspondente
    - DESCRIÇÃO: Aplicativo desenvolvido com API's do Loquendo TTS permite que o texto seja pronunciado em português com uma voz feminina, incluindo a prosódia determinada no módulo precedente.
5. Sincronia Labial
    - ENTRADA: Texto correspondente ao discurso a ser pronunciado pelo robô
    - SAÍDA: PWM para comando da posição do servo-motor que aciona o mecanismo de movimentação da boca
    - DESCRIÇÃO: Dado o texto que o robô irá pronunciar, uma sequência de comandos é enviada para o microcontrolador para que esse acione o mecanismo da boca conforme as vogais e consoantes da fala.

3.2- ROBÔ MINERVA

A parte mecânica do robô Minerva não foi desenvolvida neste trabalho, mas em trabalhos anteriores, portanto este trabalho apenas apresentará a parte mecânica com a qual se deu inicio ao trabalho. Após a apresentação da parte mecânica, serão apresentados os circuitos projetados, bem como todos os componentes eletrônicos presentes neste trabalho. Por fim, ainda neste capítulo será explicado o funcionamento do mecanismo e como este se relaciona com a parte eletrônica do projeto para ser controlada pelos softwares.

Nas figuras a seguir é possível observar a construção mecânica como um todo em diferentes ângulos do robô, com e sem a máscara:

Figura 22- Vista frontal do robô Minerva (sem a máscara do lado esquerdo e com do lado direito)

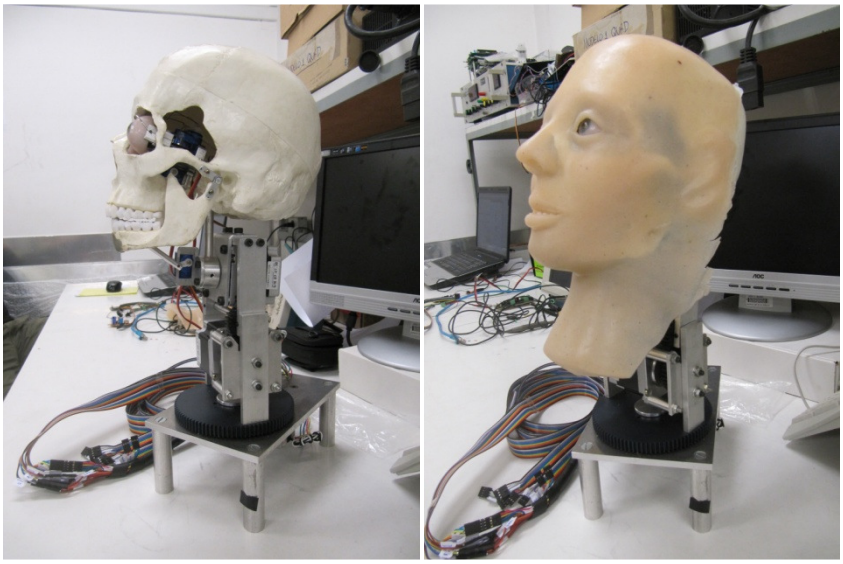

Figura 23- Vista lateral do robô Minerva, sem e com a máscara (esq. e dir. respectivamente)

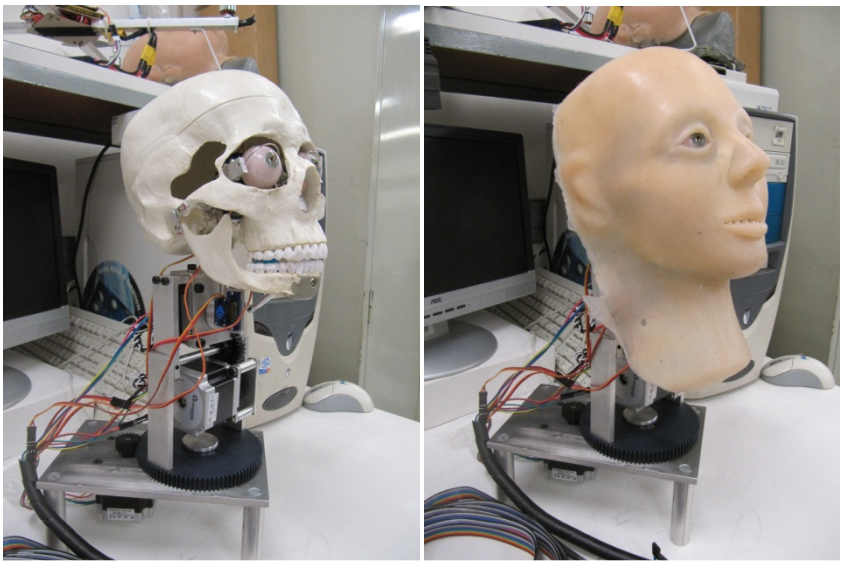

Figura 24- Vista lateral do robô Minerva, sem e com a máscara (esq. e dir. respectivamente)

Os sub-ítens a seguir explicam os detalhes de cada subconjunto:

A movimentação do robô é acionada por três motores de passo da Akiyama, modelo AK56H/3-1.8 com tensão nominal de 4.32V e corrente de 2.4 A/fase. Os três motores são responsáveis pelos três graus de liberdade de movimentação do pescoço do robô.

A seguir imagens dos motores já acoplados ao sistema:

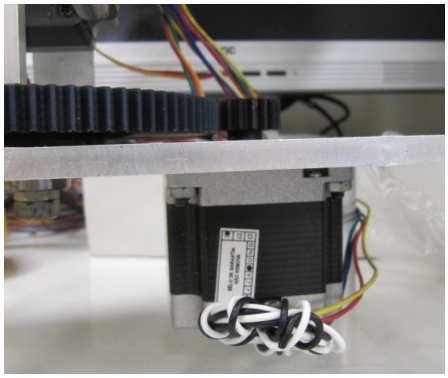

Figura 25- Motor responsável pela rotação da cabeça em torno da vertical

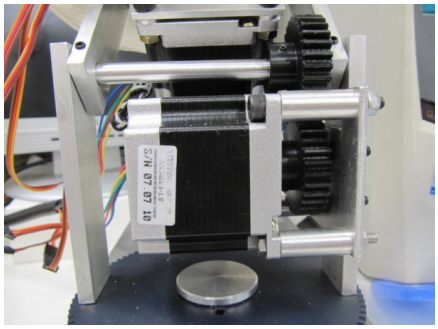

Figura 26- Motor responsável pela rotação da cabeça em torno da horizontal (movimentação de “sim”)

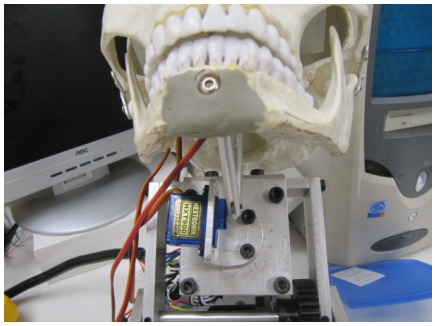

Figura 27- Vista frontal do motor responsável pela rotação da cabeça em torno do eixo perpendicular à face (movimento de "não")

Além dos três graus de liberdade da movimentação da cabeça, a estrutura ainda possui quatro servomotores da Hextronik, modelo HXT900, responsáveis pela movimentação dos elementos da face (mandíbula, pálpebras e olhos).

- **Boca:** Para a movimentação da boca (mandíbula) é utilizado um servo conectado por uma haste articulada, como pode ser observado na figura abaixo. Esta configuração permite controlar a abertura e o fechamento da boca através da movimentação da mandíbula, enquanto a parte superior permanece fixa à cabeça.

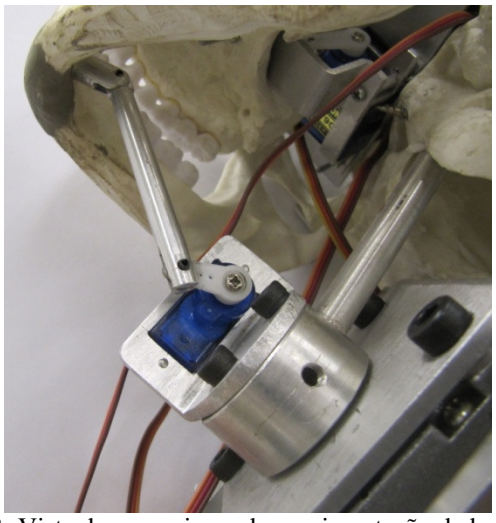

Figura 28- Vista do mecanismo de movimentação da boca (servo motor e haste articulada)

- **Pálpebras**: Para a movimentação das pálpebras é utilizado um servo motor, acoplado a um mecanismo responsável por girar o aro das pálpebras. Nas imagens abaixo é possível notar as pálpebras abertas (aro na parte superior dos olhos) e fechadas (aro posicionado no meio dos olhos). Também é possível notar o detalhe do mecanismo de movimentação e fixação do motor na cabeça. Na parte superior da figura anterior (mecanismo da boca) é possível notar o detalhe do motoro que movimenta as pálpebras.

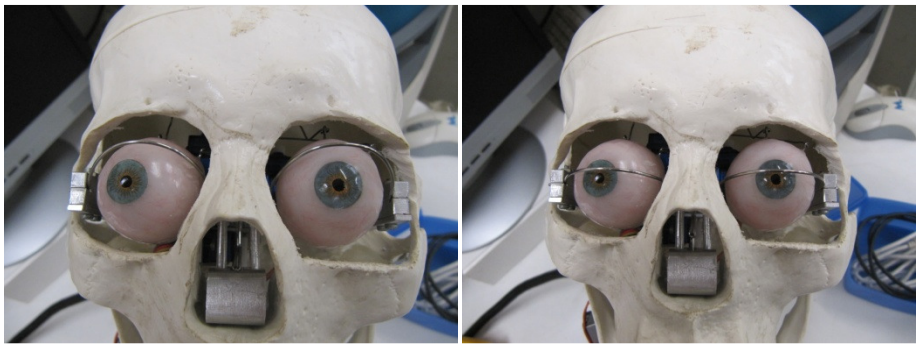

Figura 29- Detalhe das pálpebras abertas (a esq.) e fechadas (a dir.) representadas pelo aro metálico

- **Olhos**: Para a movimentação dos olhos são utilizados dois servomotores acoplados aos olhos por hastes metálicas para movimentação conjunta de ambos os olhos. Os olhos estão fixados a cabeça através de juntas esféricas. Para movimentar os olhos é necessário acionar os dois motores simultaneamente para que os olhos movimentem de forma sincronizada e com trajetórias e posições semelhantes às do olho humano. Na imagem a seguir é possível observar uma vista superior dos dois servomotores e das hastes de fixação, bem como dos olhos e das juntas esféricas de fixação.

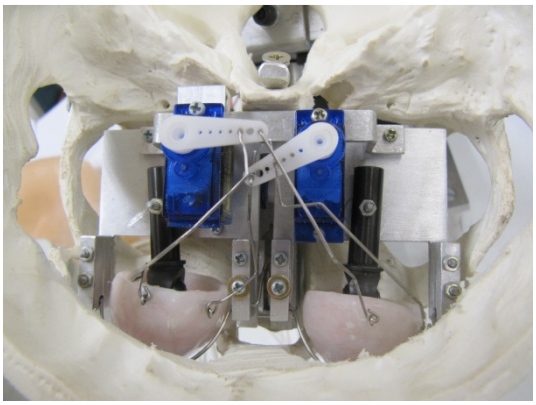

Figura 30- Vista superior dos servo-motores de controle dos movimentos dos olhos, com detalhes do mecanismo de movimentação

3.3- ARQUITETURA DE HARDWARE

Uma arquitetura de hardware foi desenvolvida para acionamento dos componentes do robô. Ela é composta dos seguintes componentes:

- Dois computadores: um contendo os módulos ASR e *chatbot* AIML e outro com o módulo TTS, que se comunica com o robô enviando os dados de movimentação labial;
- Um módulo da fabricante FTDI (*Future Technology Devices International*), para permitir a comunicação entre o computador e o microcontrolador;
- Um microcontrolador Cerberus, da GHI Electronics;

- Um módulo IO60P16, para obtenção de mais sinais de PWM;
- Um USB Client SP para programação do microcontrolador;
- Três *drivers* para acionamento dos motores de passo
- Uma placa de regulação de tensão de 24V para 5V
- Uma fonte de 24V
- Cabeamento
- Motores

O esquema da arquitetura de hardware encontra-se na Figura 31.

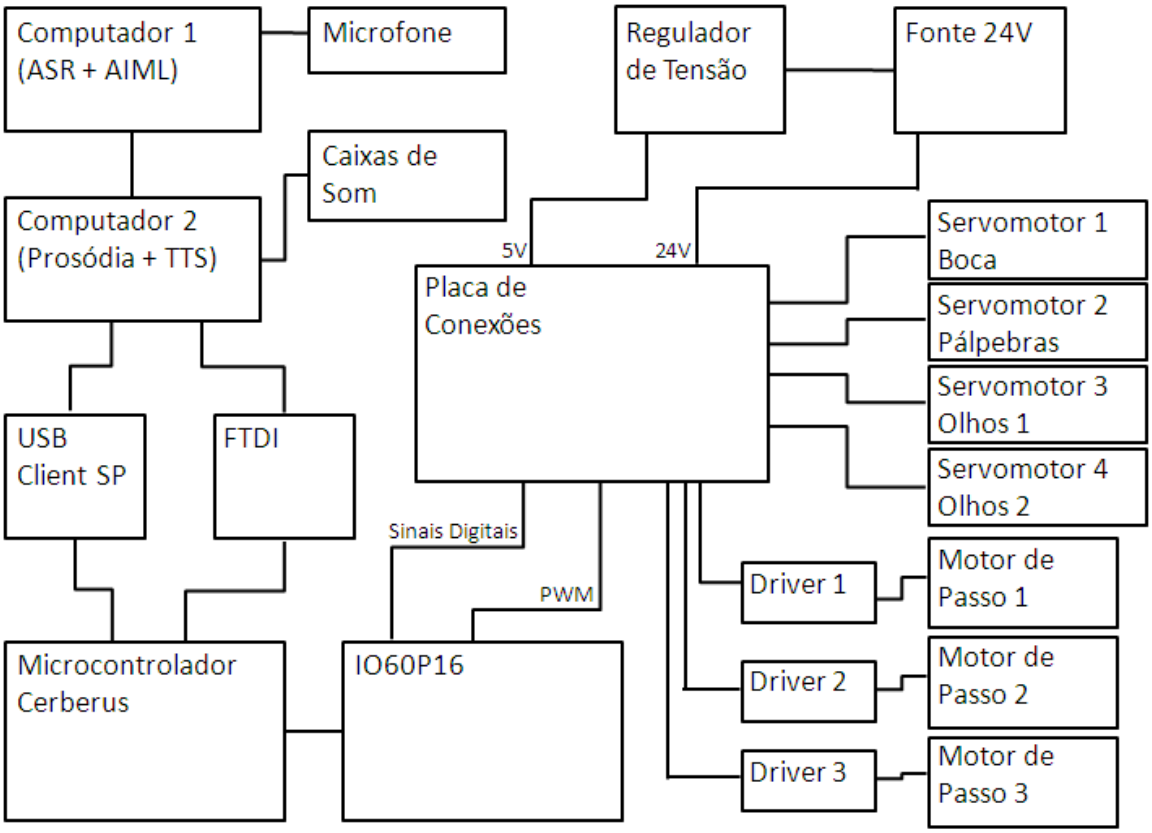

Figura 31- Esquema da Arquitetura de Hardware

3.3.1- Componentes eletrônicos utilizados:

Nesta seção são apresentados circuitos eletrônicos que não foram projetados, mas sim comprados para utilizar neste projeto.

3.3.1-1. Controle da parte mecânica

Para controlar a parte mecânica do robô, foi adotado o microcontrolador Cerberus (Figura 32), que possui como microprocessador um Cortex M4 de 32bis e 168 MHz. Este microcontrolador funciona baseado em módulos, que são necessários para fazer as conexões.

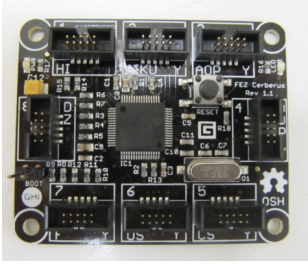

Figura 32- Cerberus – Microcontrolador utilizado no projeto

Dentre os módulos disponíveis, foram adquiridos dois deles, IO60P16 e USB Client SP, detalhados abaixo:

- **IO60P16**: É um módulo de expansão, com mais 60 pinos. Este módulo foi adquirido, pois para movimentar todos os motores da cabeça, são necessários sete sinais de PWM, e o Cerberus possui apenas seis. Este módulo possui 16 PWMs, então também é útil para o caso de expandir este projeto no futuro utilizando outros motores. A conexão entre os dois módulos é feita por um cabo mini-flat da própria fabricante. Neste projeto, por uma questão de uniformização, todos os PWMs utilizados são provenientes do módulo IO60P16.

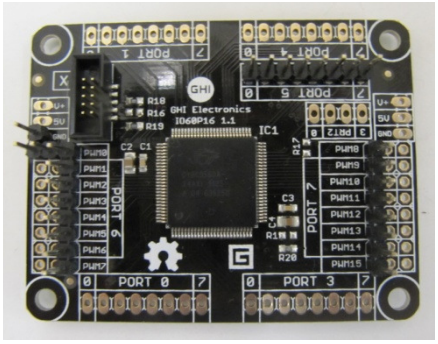

Figura 33- IO60P16 - Módulo de expansão para o Cerberus com 16 PWMs

- **USB Client SP**: Este módulo é necessário para fazer a conexão entre o Cerberus e o computador via USB, para programação e debug do software. A conexão entre os dois módulos é feita por um cabo mini-flat da própria fabricante do módulo.

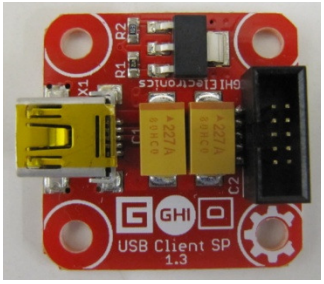

Figura 34- USB Client SP – Módulo para comunicação do computador com o Cerberus via USB

Na figura 35 é possível observar como é conexão entre os módulos citados:

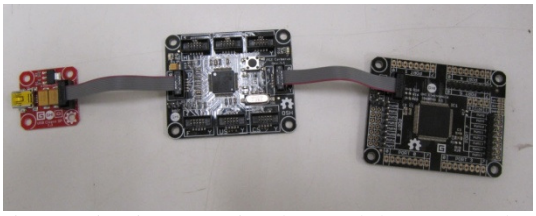

Figura 35- Vista superior dos módulos conectados

3.3.1-2. Comunicação entre o computador e o microcontrolador

Para permitir a comunicação entre o módulo de TTS, responsável pela fala e o microcontrolador, é necessária uma segunda conexão serial entre o microcontrolador e o computador. Para esta conexão foi utilizado um componente da FTDI para fazer a conversão de serial para USB.

Foi feito um cabo para conectar as saídas TX, RX e GND do soquete 2 do Cerberus com o FTDI para permitir esta comunicação.

Na figura 36 o módulo FTDI utilizado.

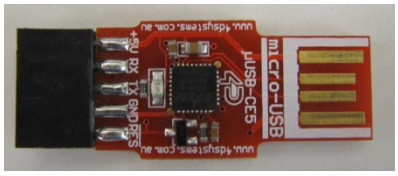

Figura 36- Utilizado para conexão entre o Cerberus e o computador

3.3.1-3. Controle dos motores de passo

No controle dos motores de passo[2], são utilizados *drivers* da Akiyama, modelo AKDMP16-4.2A, que são opto isolados para garantir a segurança do circuito eletrônico.

Os *drivers* recebem dois sinais digitais para indicar habilitação e direção de rotação, bem como o sinal de PWM para garantir a movimentação.

Os *drivers* precisam ser configurados, de acordo com o nível de corrente dos motores, bem como o número de passos por revolução desejados. Este ajuste é feito através de chaves presentes no próprio *driver*, seguindo o manual do *driver*.

[2] Até o fechamento e entrega deste trabalho, os motores de passo não haviam sido utilizados em conjunto, o foco permaneceu no controle do servo responsável pela movimentação da boca para garantir a sincronização com a fala.

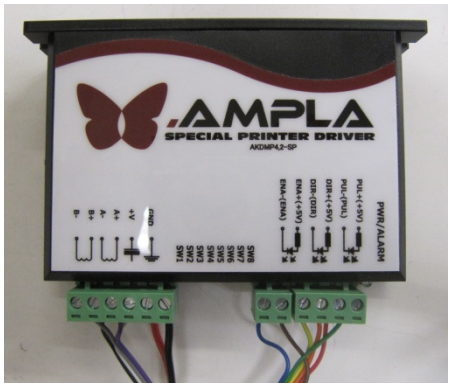

Figura 37- AKDMP16 -4.2A – *Driver* para motor de passo (Akiyama)

3.3.2- Circuitos eletrônicos projetados:

Nesta seção são mostrados os circuitos eletrônicos que foram projetados e construídos para a utilização neste projeto.

Para conectar a parte mecânica e os componentes eletrônicos utilizados, foi necessário projetar alguns circuitos eletrônicos auxiliares apresentados a seguir.

A parte eletrônica do robô, como visto no item anterior, utiliza tensão de alimentação de +5V tanto para o controlador quanto para os servo motores. Enquanto que a parte mecânica (motores de passo) utilizam uma tensão de +24V. Desta forma, viu-se necessário utilizar um circuito para regular a tensão da fonte de +24V para que mesma gerasse também +5V, utilizando desta maneira a mesma fonte para ambas tensões de alimentação.

Foi utilizado um regulador de alta tensão (HV) ajustável, montado segundo o seguinte esquema de ligação na Figura 38.

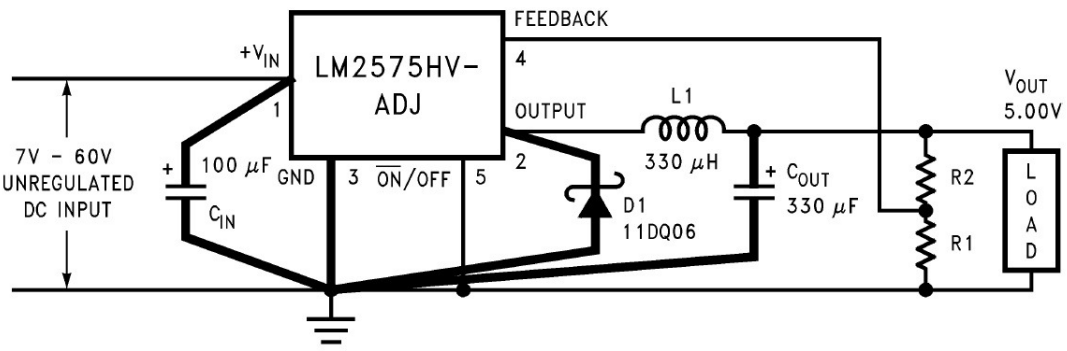

Figura 38- Esquema de ligação do regulador de tensão ajustável (NATIONAL SEMICONDUCTORS, 2004)

A entrada deste regulador permite até 60V, o que é suficiente para esta aplicação, porém para que a saída seja de acordo com o necessário (+5V) é necessário ajustar as duas resistências (R1 e R2) seguindo as Equações 2 e 3.

$$V_{OUT} = V_{REF}\left(1 + \frac{R2}{R1}\right) \quad (2)$$

$$R2 = R1\left(\frac{V_{OUT}}{V_{REF}} - 1\right) \quad (3)$$

Foi assumido 1680 ohms para R2 (obtidos através de duas resistências de 1500 e 180 ohms ligadas em série). Para uma tensão de saída de +5V, sabendo que a tensão de referência é de 1,23V (de acordo com o *datasheet* do regulador de tensão LM2575HV) tem-se uma resistência R1 necessária de aproximadamente 550 ohms.

Porém por ajustes necessários, a resistência final para R1 utilizada foi de 526 ohms (obtida através de duas resistências ligadas em série, de 470 ohms e 56 ohms). A imagem do circuito após a montagem pode ser observada na seção de resultados.

Além do circuito regulador de tensão foi desenvolvido um circuito auxiliar para concentrar todas as ligações eletrônicas. Foi adotada esta abordagem para ser possível a utilização de um cabeamento estruturado para as conexões da cabeça. Assim toda a parte eletrônica fica concentrada e distante da cabeça, além de facilitar o transporte da cabeça independentemente da parte eletrônica, e permitindo também que a cabeça possa ser alocada a distância da eletrônica.

Esta abordagem permite que trabalhos futuros sejam feitos com a cabeça do robô, sem a preocupação de encaixar todos os componentes eletrônicos. A manutenção fica mais simples também uma vez que basta soltar os cabos e as partes (eletrônica, mecânica) ficam separadas.

Na placa de conexões ainda foram adicionados dois LED´s, um vermelho para indicar que o circuito está sendo alimentado com +24V e um verde, para indicar que o circuito também está sendo alimentado com +5V.

A seguir, na Figura 39, a imagem da placa de conexões, seguida do esquema de ligações da mesma, e na Figura 40 é possível observar como ficou o cabeamento estruturado.

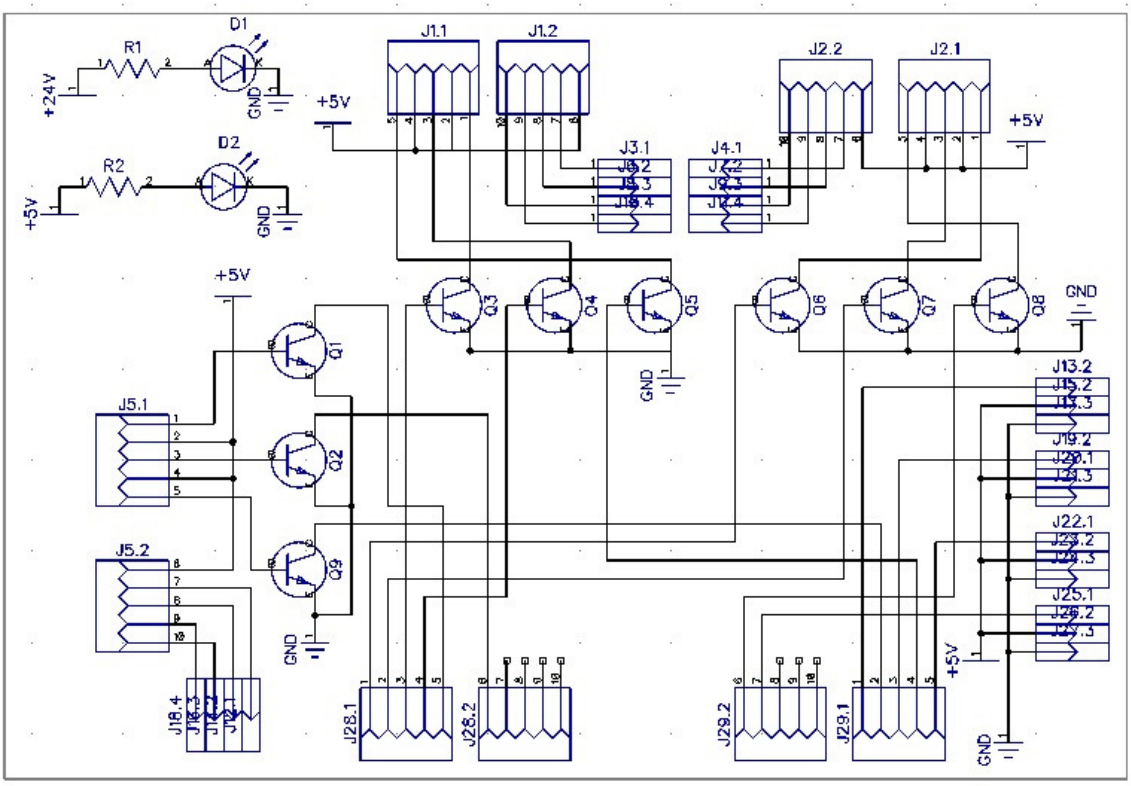

Figura 39- Esquemático das ligações da placa de conexões

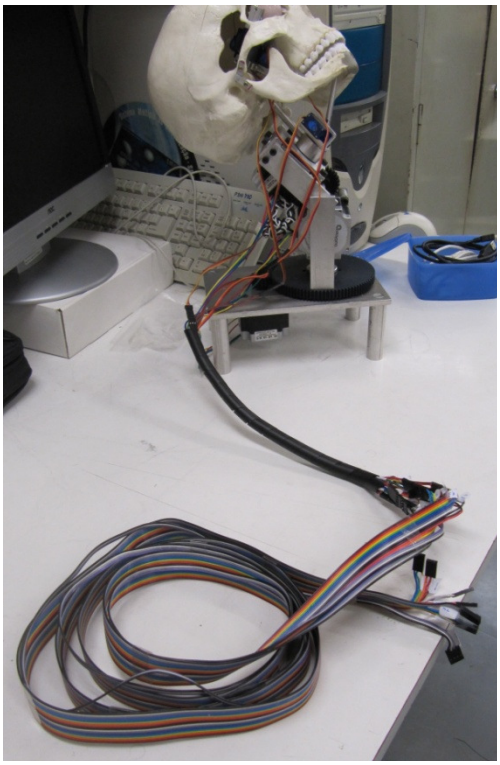

Figura 40- Vista da cabeça conectada ao cabeamento estruturado

3.3.3- Funcionamento do mecanismo da boca com a parte eletrônica

Neste projeto o principal mecanismo de movimentação é o da boca, responsável pela sincronização da fala. Este mecanismo não foi projetado neste trabalho, mas em trabalhos passados. Na Figura 41 é possível notar os limites de operação do mecanismo de movimentação da boca.

Da abertura completa até o fechamento completo da boca, tem-se uma rotação de cerca de 90 graus do servo motor, o que foi mapeado e implementado no software, através dos valores limite para largura do pulso do PWM, conforme será explicado mais detalhadamente nas seções seguintes. Serão explanados também quais os critérios utilizados para cada letra / abertura da boca.

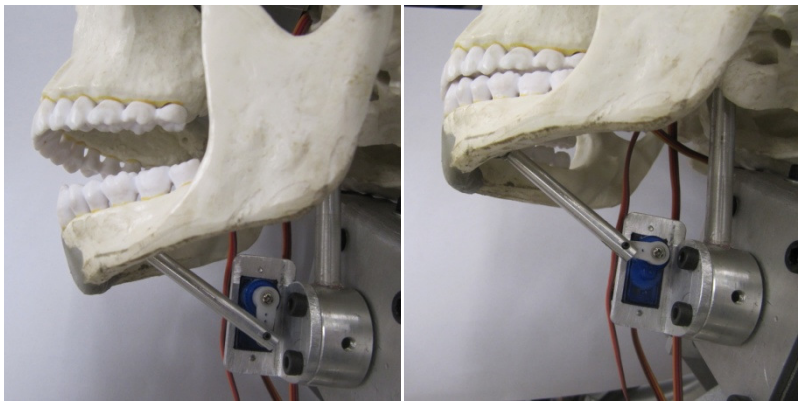

Figura 41- Vista lateral da movimentação da boca (aberta, a esq. e fechada, a dir.)

3.4- ARQUITETURA DE SOFTWARE

O desenvolvimento do software de conversação para o robô foi dividido em dois computadores, no primeiro foi desenvolvido o módulo de ASR integrado ao *chatbot* AIML. No outro computador foi desenvolvido o módulo TTS, com geração de prosódia e comunicação com o hardware para comando de movimentação da boca. A opção pela utilização de dois computadores foi feita porque só foi disponibilizada uma licença para o Loquendo ASR e uma para o Loquendo TTS. Instalando uma licença em cada computador permite o trabalho em paralelo, o que agiliza o desenvolvimento. Entretanto, para projetos futuros, o software pode ser unificado e instalado numa única máquina.

Esta seção tem como objetivo apresentar os softwares desenvolvidos, começando pelo módulo ASR, seguindo para *chatbot* AIML. Na sequência é apresentado o módulo TTS e o módulo de sincronia labial, destacando o protocolo utilizado de comunicação.

3.4.1- Módulo ASR

O desenvolvimento de um reconhecedor automático de fala é complexo, pois envolve definição das características de áudio que serão extraídas, manipulação dessas características através de modelos estatísticos e treinamento do classificador. Como este trabalho não tem como objetivo desenvolver um novo reconhecedor de fala, mas sim demonstrar uma arquitetura de gerenciamento de diálogos, optou-se pela utilização de uma solução comercial para esse problema. A solução escolhida foi o software Loquendo ASR, que possui um pacote para utilização em língua portuguesa que não necessita de treinamento posterior. O Loquendo ASR é disponibilizado pela empresa Nuance que fornece soluções de tecnologia na área de fala com aplicações em diversas indústrias, como indústria de telefonia, indústria automotiva e bancos (NUANCE, 2012).

O software do módulo ASR, desenvolvido utilizando as API's do Loquendo, tem as seguintes funcionalidades:

- Configurar o reconhecimento de voz, através de seleção de gramática, configuração de parâmetros e modos de reconhecimento;
- Comandar o reconhecimento de voz;
- Mostrar os resultados do reconhecimento numa interface;
- Enviar a frase reconhecida ao *chatbot* AIML e receber sua resposta;
- Enviar a resposta do *chatbot* ao outro computador, para que possa ser utilizado para fala do robô.

As classes utilizadas para desenvolvimento do software, conforme o diagrama da Figura 42 são:

- Controller: é a classe principal, responsável pelo gerenciamento de todas as funcionalidades do software. Nesta classe o objeto de reconhecimento os métodos startRecog() e stopRecog() comandam o início e fim do ciclo de reconhecimento. O método grammarCompile() compila e carrega uma nova gramática no reconhecedor. Por fim, o método updateResults() é chamado internamente para tratar o resultados do reconhecimento, chamando métodos das outras classes para adquirir resposta do AIML, enviar a resposta ao outro computador e escrever o diálogo na interface.
- ASRInstance: esta é a classe do Loquendo ASR responsável pelo reconhecimento de voz. Através dos métodos setParameter(), getParameter() e setRecogMode() podem ser configurados os parâmetros e o modo de

reconhecimento utilizado. O método recog() realiza o reconhecimento, devolvendo os resultados como um objeto da classe ASRRecRes.

- ASRRecRes: contém os resultados do reconhecimentos, como a lista de hipóteses, o número de hipóteses, o SNR e o aviso de rejeição.
- ASRHypo: é a representação de cada hipótese de frase para o discurso proferido. Os métodos getText() e getConfidence() retornam respectivamente o texto da hipótese e o seu nível de confiança
- Interface: esta classe é responsável por criar a interface visual de interação do usuário com o sistema. Os métodos permitem o carregamento de arquivos com a gramática, a apresentação das linhas de diálogo e dos *status* de detecção ao usuário e a interação através de botões e caixas de seleção.
- BotAiml: cria o *chatbot* utilizando o arquivo de propriedades do AIML e retorna a resposta do robô através do método getResponse(speech:String).
- TextOutput: responsável por escrever o arquivo SSML contendo a resposta do robô a ser pronunciada através do método speak(response:String). O arquivo é escrito no outro computador, utilizando a rede local.

Na Figura 42 está representado o diagrama de classes do módulo ASR, com alguns dos atributos e os principais métodos utilizados.

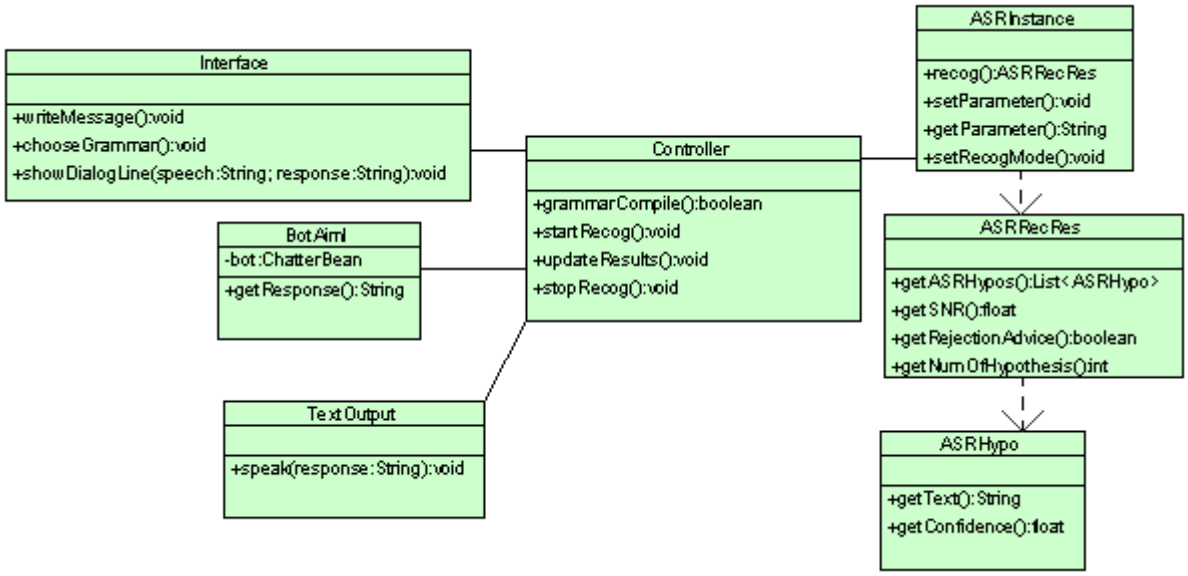

Figura 42- Diagrama de Classes do Módulo ASR

Na Figura 43 está representado o caso de uso principal do programa, o reconhecimento de voz. A sequência inicia com o comando startRecog() feito pelo usuário através da interface. Esse comando ativa o controlador que realizará o seguinte laço:

- Aciona o método recog() da instância do ASR, que aguardará que um discurso seja reconhecido na fonte de áudio e em seguida retornará os resultados do reconhecimento.
- Captura a resposta do *chatbot* chamando o método getResponse() da classe BotAiml
- Escreve o arquivo com a resposta do robô no outro computador através do método speak() da classe TextOutput
- Aciona o método showDialogLine() para apresentar ao usuário o resultado da detecção do seu discurso e a resposta dada pelo robô a essa proposição

O controlador permanece nesse laço até que o usuário, através da interface, chame o método stopRecog().

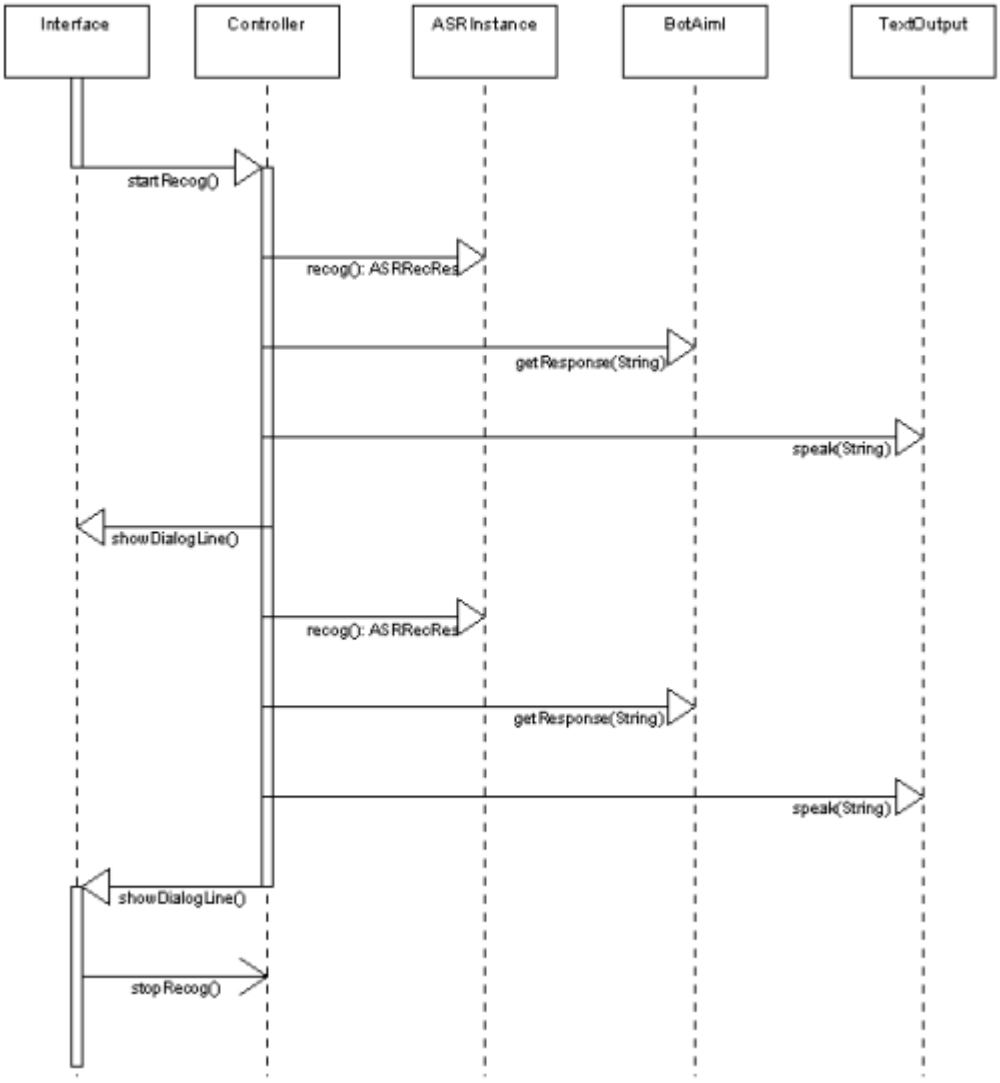

Figura 43- Diagrama de Sequência do Caso de Uso de Reconhecimento

3.4.2- Chatbot AIML

A configuração do *chatbot* é feita através dos arquivos properties.xml, context.xml e substitutions.xml, conforme visto na seção 1.2.

O arquivo properties.xml é o que determina onde estão todos os outros arquivos do *chatbot*. É através dele que o módulo ASR cria o objeto *bot*, o qual é capaz de responder às proposições segundo as categorias dos arquivos AIML.

As categorias do AIML não constam em um único arquivo, elas estão separadas em oito arquivos AIML diferentes. A divisão por arquivos é feita por funcionalidade, por exemplo, um arquivo é responsável pelo conhecimento do robô, outro por sua personalidade e outro por conter cumprimentos e despedidas.

Mais detalhes sobre a implementação do *chatbot* são dadas no capítulo de resultados e discussões, na seção 4.2.

3.4.3- Módulo TTS

O desenvolvimento de um software de conversão de texto em fala é complexo, pois envolve tratamento e composição de sinais, entre outras dificuldades. Como este trabalho não tem como objetivo desenvolver um novo conversor de texto em fala, mas sim demonstrar uma arquitetura de gerenciamento de diálogos, optou-se pela utilização de uma solução comercial para esse problema. A solução escolhida foi o software Loquendo TTS, que já possui uma série de API´s para a conversão do texto em fala, inclusive com uma voz brasileira.

O software criado para o módulo de fala foi desenvolvido utilizando-se as API's do Loquendo TTS, tendo as seguintes funcionalidades:

- Configurar a voz que será utilizada, através da configuração de parâmetros da voz;
- Controlar a conversão do texto, selecionando o formato do arquivo de entrada (salvo pela AIML) e a saída (placa de som do computador ou arquivo de áudio);
- Converter o texto e gerar o som, ou gravar o arquivo, de forma síncrona ou assíncrona;
- Tratar a resposta enviada pelo AIML e converter o texto em comandos a serem enviados ao microcontrolador;
- Enviar os comandos de movimentação da boca para o microcontrolador.

As classes utilizadas para desenvolvimento do software são:

- TTS: é a classe principal, responsável pelo gerenciamento de todas as funcionalidades do software. Nesta classe a sessão nova é instanciada e o objeto de conversão "Reader" do TTS é instanciado. Ele é configurado com a voz a ser utilizada e com o texto que será convertido em formato SSML e o resultado da conversão é direcionado para a placa de som do computador.

  Esta classe também implementa o tratamento da texto recebido para convertê-los em comandos a serem enviados para o microcontrolador. A conversão é feita através do método convertSSML() e os resultados são enviados para a placa de som do computador através do método read(). O envio dos comandos para o microcontrolador é feito pelo método sendMessage().
- ReadWriteTextFiles: Permite a escrita e leitura de arquivos de texto.

3.4.4- Comunicação Serial

A comunicação serial utilizada para transmitir os módulo TTS para o microcontrolador foi via USB, utilizando-se um chip FTDI responsável por fazer a conexão com o computador, conforme visto na seção 3.3.1-2 com detalhes do hardware.

Apesar da possibilidade de execução, neste trabalho não foi elaborada nenhum protocolo mais complexo de comunicação. É feita apenas a transmissão do valor da largura de pulso de cada letra pelo módulo TTS e recebida pelo microcontrolador, que faz a interpretação e utilização do comando para gerar o PWM necessário para cada letra em cada posição da boca, e que será mais detalhado na seção seguinte.

3.4.5- Módulo de Sincronia Labial

Esta etapa do trabalho é uma aproximação realizada assumindo-se a estrutura mecânica resultado de um projeto anterior a este. Portanto este trabalho não apresenta a sincronia dos lábios com a fala, uma vez que para tornar esta sincronização possível, seriam necessários mais graus de liberdade e motores para controlar os lábios propriamente ditos, mas sim a sincronização da boca com a fala, utilizando-se o único grau de liberdade disponível.

O módulo de sincronia labial consta de um software responsável pelo controle dos motores, tanto dos servos quanto dos motores de passo. Este software também recebe via serial os comandos enviados pelo módulo TTS, que envia os valores de largura de pulso a serem utilizados pelo PWM de controle da boca.

Os valores de largura de pulso são gerados pelo próprio software feito para o TTS, conforme explicado na seção 3.4.3 e detalhado no capitulo 4 de resultados, na seção 4.5.

O módulo de sincronia labial possui apenas duas classes, uma que contém os módulos utilizados e que é criada automaticamente pelo programa Gadgeteer ao conectarem-se os módulos visualmente, como será explicado com mais detalhes na seção 4.5 do capitulo de resultados, e uma segunda classe que é a principal, onde todo o programa foi desenvolvido.

A classe principal contem as interrupções, faz o instanciamento das variáveis utilizadas, possui também um método para tratar os eventos da comunicação serial e um método de controle dos PWM's, a qual modifica e envia os novos valores de PWM toda vez que é chamada dentro da interrupção.

4- RESULTADOS E DISCUSSÕES

4.1- MÓDULO DE RECONHECIMENTO DE FALA - ASR

A interface do aplicativo desenvolvido para o módulo ASR pode ser vista na Figura 44. No painel superior são mostrados os resultados da detecção, em azul está a melhor hipótese, ou hipótese número 1. Em verde está o nível de confiança dessa hipótese. As demais hipóteses, de 2 até 5, são listadas em seguida, na cor preta. A janela à direita é uma Janela Log, cuja função é mostrar informações gerais sobre o andamento do programa e outros resultados de detecção como: Rejeição e SNR.

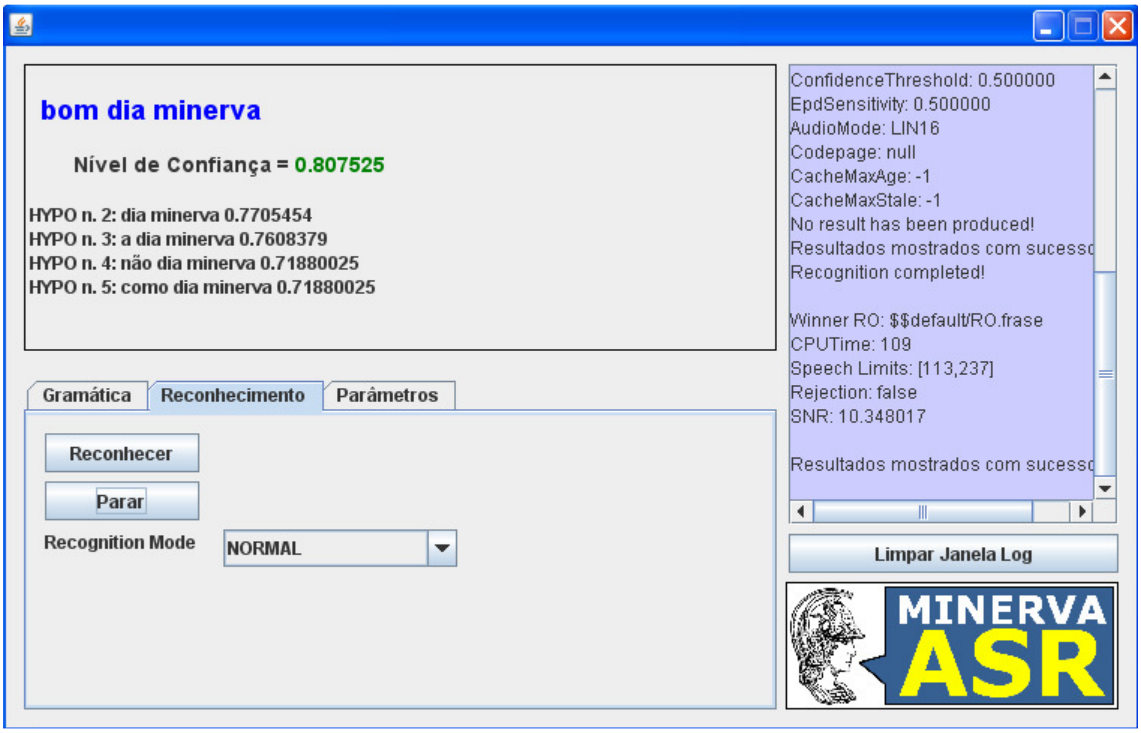

Figura 44 – Interface do aplicativo de testes desenvolvido

O painel inferior possui três guias. A primeira é Gramática, onde há um botão que permite selecionar um arquivo no computador com a gramática. A segunda guia do painel é Reconhecimento, na qual se pode ativar o reconhecimento da frase, e também selecionar um dentre os cinco modos de reconhecimento: *normal*, *silent*, *semisilent*, *silent_restart* e *semisilent_restart*.

A última guia do painel inferior é a guia Parâmetros (Figura 45). Nela é possível alterar alguns parâmetros da instância de reconhecimento.

Figura 45 – Guia Parâmetros

Quanto à gramática utilizada neste trabalho, não foi possível dividir as palavras em classes, uma vez que tem-se o objetivo de falar sobre coisas em geral, e para tanto as possibilidades de frases a serem criadas são muito numerosas. A gramática utilizada no trabalho foi, portanto, conforme o exemplo da Figura 46.

```
#ABNF 1.0 UTF-8;

meta "loq-remark" is "frase";
meta "loq-globmodel" is "DEFAULT";

language  pt-br;
mode  voice;
root  $frase;
tag-format <semantics/1.0>;

$frase = (  $words |
            $words  $words |
            $words  $words  $words);

 $words = (
                  a |
                  abaixo |
                  abelhas |
                  .
                  .
                  .
                  zangado |
                  zero |
                  zombando
                );
```

Figura 46- Gramática semelhante à que foi utilizada neste trabalho

Nesse caso só existe uma classe de palavras, chamada *words* e as frases são sequências dessas palavras, neste caso de no máximo três palavras, mas nos testes do robô foram admitidas frases de até seis palavras. O tamanho final do vocabulário ficou em 3159 palavras, que são todas as palavras que possuem alguma utilização pelo *chatbot* AIML, como será apresentado na seção 4.3.

4.2- TESTES COM O MÓDULO ASR

Com o intuito de compreender melhor o desempenho do reconhecimento de voz realizado através do Loquendo ASR foram realizados alguns testes. Os resultados de cada um destes testes são mostrados nesta seção, seguidos pela configuração final do ASR.

4.2.1- Teste de tamanho de vocabulário

No primeiro teste foi testada a eficácia do reconhecimento do ASR quando se utilizam grandes vocabulários, ou seja, vocabulários com milhares de palavras. Deseja-se que o robô seja capaz de conversar sobre assuntos gerais, portanto é necessário que este seja capaz de reconhecer o maior número de palavras. Para realizar esse teste primeiramente foi gerado um conjunto de 50000 palavras diferentes da Língua Portuguesa. Esse conjunto foi extraído a partir de um conjunto de livros, coletando todas as palavras e excluindo as palavras repetidas. A lista dos livros utilizados nessa atividade consta na Tabela 2.

Tabela 2 – Livros dos quais foram extraídas palavras

Título	Autor
Carolina	Casemiro de Abreu
A viúva sobral	Machado de Assis
As religiões do Rio	João do Rio
As mentiras que os homens contam	Luís Fernando Veríssimo
Crônicas	Autor Desconhecido
A Escrava Isaura	Bernardo Guimarães
Coração, cabeça e estômago	Camilo Castelo Branco
O que é o casamento?	José de Alencar
Todas as histórias do analista de Bagé	Luís Fernando Veríssimo
A Mão e a Luva	Machado de Assis
Dom Casmurro	Machado de Assis
O Primo Basílio	Eça de Queirós
A Moreninha	Joaquim Manoel de Macedo
A filosofia entre a religião e a ciência	Bertrand Russel
A cidade do Sol	Tommaso Campanella
A Dança dos Ossos	Bernardo Guimarães
Várias Histórias	Machado de Assis
Jean-Jacques Rousseau	Michel Soëtard
Paulo Freire	Celso de Rui Beisiegel

Com esse conjunto de 50000 palavras foram criadas 6 gramáticas, com números diferentes de palavras em cada. Os números foram 60, 500, 1000, 5000, 20000 e 50000. As palavras foram escolhidas aleatoriamente, para evitar excesso de palavras foneticamente

parecidas no mesmo vocabulário, como por exemplo: caminho e caminhe. Não foi aplicada nenhuma regra gramatical da língua. As frases foram consideradas simplesmente como sequências de uma, duas ou três palavras quaisquer, conforme o exemplo da seção 2.1.

Quarenta frases foram usadas no teste, vinte delas com apenas uma palavra e vinte com três palavras. Os testes consistiram em proferir cada frase cinco vezes, anotando o número de vezes que o reconhecedor acertava completamente o que foi dito. Até mesmo erros de gênero e pluralidade, que são mais comuns, foram considerados para descartar um acerto.

Usar vinte frases diferentes foi uma opção para aumentar a variedade de palavras reconhecidas, fugindo de erros causados por escolhas de frases muito fáceis ou muito difíceis de reconhecer. A repetição de cada frase cinco vezes foi uma estratégia para diminuir a influência do vício do falante, que, percebendo os erros que o reconhecedor comete, começa a proferir de forma mais pausada e falando mais alto, para facilitar o reconhecimento.

A Tabela 3 mostra os resultados da primeira bateria dos primeiros testes.

Tabela 3 – Resultados dos testes de reconhecimento de palavras

		60	500	1000	5000	20000	50000
1	Bom	5	5	5	5	5	5
2	Sim	5	5	5	5	5	3
3	Casa	5	5	5	5	4	3
4	Maçã	5	4	5	4	3	0
5	Mundo	5	5	5	5	3	3
6	Útil	5	5	5	3	2	0
7	Riso	5	5	5	5	4	5
8	Quebrar	5	5	5	5	3	4
9	Noite	5	5	5	5	4	3
10	Missão	5	5	5	5	5	2
11	Fazenda	5	5	5	5	5	5
12	Honesto	5	5	5	2	0	0
13	Sábado	5	5	5	5	5	5
14	Promessa	5	5	4	4	5	4
15	Escuro	5	5	5	4	5	2
16	Biblioteca	5	5	5	5	5	5
17	Injustiça	5	5	5	5	3	1
18	Homenagem	4	5	5	5	5	5
19	Termômetro	5	5	3	4	4	5
20	Paralelepípedo	5	5	5	5	5	5
	MÉDIA	4,95	4,95	4,85	4,55	4,00	3,25
	DESVIO	0,22	0,22	0,49	0,83	1,34	1,86
	MÉDIA (%)	99%	99%	97%	91%	80%	65%
	DESVIO (%)	4%	4%	10%	17%	27%	37%

Nessa primeira bateria o teste foi feito com palavras isoladas, com um vocabulário de até 5000 palavras a média de acertos por palavra foi superior a 4,5, ou seja, mais de 90% das palavras proferidas foram identificadas corretamente. A partir de 20000 palavras este nível de acerto vai caindo rapidamente, primeiro para 80% e depois para 65%, com um vocabulário de 50000 palavras. Na Tabela 3 estão os dados do teste realizado.

Na segunda bateria do primeiro teste foram usadas frases de três palavras. Neste caso, as médias de reconhecimento foram menores que no teste anterior, o que mostra que quanto mais palavras na frase, mais complicado para o reconhecedor, pois o número de possibilidades cresce. No caso de palavras isoladas, as confusões são causadas quando há duas palavras com sonoridades parecidas na gramática. Já no caso de frases, para qualquer palavra da frase pode haver outra com sonoridade parecida na gramática. Além disso, há os casos em que duas palavras, quando ditas em seguida, formam a sonoridade de outra, como por exemplo, “carro” e “céu”, que juntas formam “carrossel”.

Um exemplo interessante desse último caso é o da frase: “Bom dia Minerva”. Esta frase confunde o reconhecedor quando está sendo usada a gramática de 50000 palavras, pois nesta gramática existem as palavras “bonde” e “ia”, e o reconhecedor acaba se confundindo entre as frases: “Bom dia Minerva” e “Bonde ia Minerva”. Para nós humanos é simples saber qual a frase correta, analisando o sentido e o contexto, entretanto para o computador, que analisa somente o sinal sonoro, é muito difícil distinguir.

A Tabela 4 contém os resultados do teste com frases de três palavras. Nota-se que neste caso pouco mais da metade das amostras foi reconhecida corretamente na gramática de 50000 palavras. Com vocabulários menores, de até 5000 palavras, o acerto foi de mais de 85%.

Uma observação importante é que o desvio padrão, em ambos os testes, cresceu com o aumento do vocabulário. A explicação para isso é que em geral, quando há palavras que causam confusão no reconhecedor, pouco adianta falar mais pausado e claro, o erro persiste. Dessa forma, nos testes de vocabulário menor o desvio padrão é baixo, pois os acertos são em geral 4 ou 5. Já em vocabulários grandes, quando há dificuldades numa determinada frase, dificilmente acerta-se 1 ou 2 vezes, e no caso de não haver confusão no reconhecedor, acerta-se 4 ou 5. Essa diferença de valores faz o desvio padrão crescer.

Os gráficos da Figura 47 mostram os resultados finais do teste. É possível comparar os desempenhos em frases de uma e três palavras. Vê-se como o aumento do tamanho do vocabulário e o aumento do tamanho da frase dificultam o reconhecimento correto. O teste foi útil para compreender as potencialidades do Loquendo ASR.

Tabela 4 – Resultados dos testes de reconhecimento em frases de três palavras

		60	500	1000	5000	20000	50000
1	bom dia minerva	5	5	5	3	4	2
2	sim mãe claro	5	3	5	5	4	3
3	domingo casa cheia	5	5	5	5	5	5
4	outra maçã podre	3	5	5	5	5	1
5	mundo pequeno este	5	5	5	4	1	4
6	ferramenta útil demais	5	5	5	4	5	5
7	riso muito engraçado	5	5	3	3	3	0
8	quebrar tudo aqui	5	4	5	4	0	0
9	que noite fria	5	4	5	2	1	0
10	tenho missão cumprida	5	5	5	5	5	2
11	vendeu fazenda grande	5	5	5	5	4	5
12	sou honesto sempre	3	2	2	1	0	1
13	sábado quente hoje	5	5	5	5	5	5
14	promessa esquecida novamente	5	5	5	5	5	4
15	quarto escuro medonho	5	5	5	5	5	1
16	velha biblioteca silenciosa	5	5	5	5	5	5
17	cometi injustiça terrível	5	5	5	5	5	1
18	ganhou homenagem linda	5	5	5	5	5	2
19	comprei termômetro quebrado	4	5	2	4	4	4
20	colocou paralelepípedo novo	4	5	5	5	4	5
	MÉDIA	4,70	4,65	4,60	4,25	3,75	2,75
	DESVIO	0,66	0,81	0,99	1,16	1,77	1,94
	MÉDIA (%)	94%	93%	92%	85%	75%	55%
	DESVIO (%)	13%	16%	20%	23%	35%	39%

Viu-se, portanto, que não é possível conversar com o robô com um vocabulário tão extenso quanto o de 50000 palavras e uma gramática tão simples quanto a que foi usada. Entretanto, como o conjunto de 50000 palavras foi extraído de 19 livros, percebe-se que, mesmo com um conjunto menor de palavras, em torno de 20000, já haveria palavras suficientes para escrever livros, quanto mais para conversar sobre assuntos gerais. No caso da conversação com o robô utilizando AIML percebeu-se, entretanto, que só seria necessário colocar no vocabulário do ASR as palavras contidas nas *tags* <pattern> do AIML, já que a detecção de palavras fora do que o robô consegue responder não tem utilidade. Utilizando essa estratégia o tamanho do vocabulário ficou menor que 3200 palavras, o que, de acordo com os testes produzem resultados suficientes, mesmo que o número de palavras na frase seja aumentado.

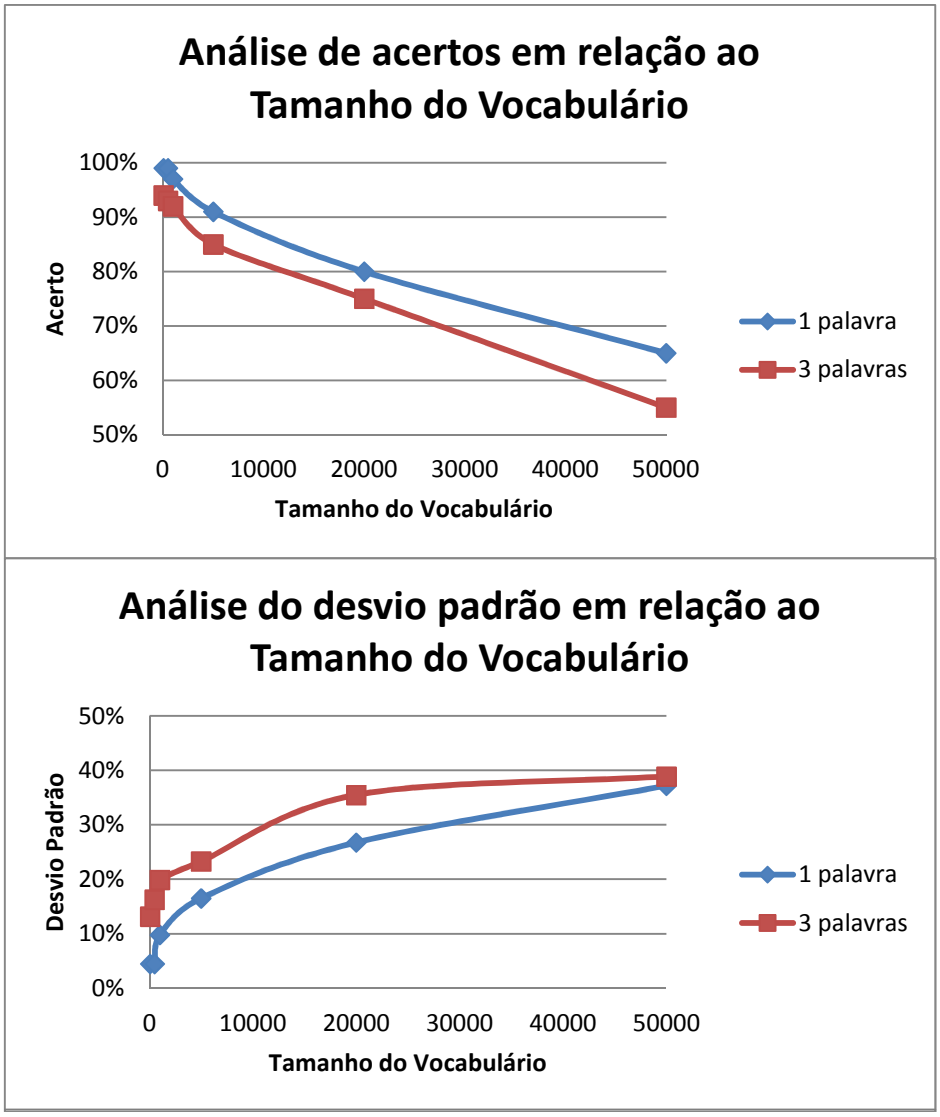

Figura 47 – Gráficos dos testes de tamanho de vocabulário

4.2.2- Teste de detecção de silêncios

Tendo como referência os dados desses testes desejou-se desenvolver uma estratégia para que se taxas maiores de detecção fossem atingidas. A primeira ideia foi detectar os silêncios nas frases, que supostamente corresponderiam ao intervalo entre uma palavra e outra. Essa ideia, entretanto foi descartada porque, ao analisar o sinal sonoro percebemos alguns problemas com essa abordagem. Na Figura 48 está o sinal de áudio da frase: “Hoje o dia está bastante ensolarado”. Note que há 8 silêncios, que dividem o sinal em 7 intervalos, entretanto a frase possui apenas 6 palavras.

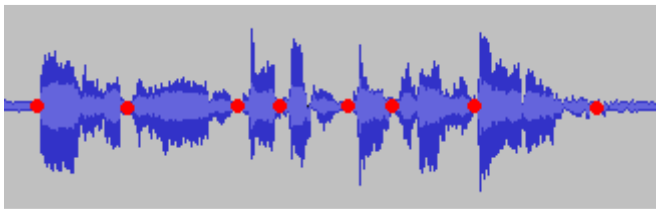

Figura 48 - Sinal sonoro da frase "Hoje o dia está bastante ensolarado."

Analisando cada trecho é possível descrever as secções como: “Hoje o”, “dia est”, “tab”, “as”, “tan”, “te enso”, “larado”. O principal problema ocorre quando uma palavra iniciada em vogal sucede outra que termina com vogal, como por exemplo: “Hoje o”, “dia está” e “bastante ensolarado”. Nesses casos não existe silêncio entre as palavras. Outro problema é que muitas vezes há silêncios dentro da própria palavra, como no som de “s” e de “n” da palavra “bastante”.

4.2.3- Teste para identificação de erro de “reset”

Mesmo com essa dificuldade decidiu-se fazer outro teste, a fim de identificar a possibilidade de problemas no “reset” entre uma palavra e outra. Usando apenas a gramática de 5000 palavras foram gravadas frases de 10, 5, 3 e 1 palavra. Anotou-se o número de palavras corretamente identificadas (ignorado artigos e outros monossílabos de som fracos). Também foi anotada a localização dos acertos e dos erros, para detectar eventuais erros de reset. O resultado deste teste consta na tabela seguinte.

Tabela 5- Teste para identificação de erro de "reset"

Frases	Acertos	Localização - Erros
10 palavras		
1-Aquele rapaz vestido de branco é o novo aluno de medicina	9	6 - é
2-O padre que visitou nossa paróquia fazia lindos sermões sobre família	10	
3-Meu avô contava muitas histórias de quando morava na fazenda	10	
4-Um homem estranho estava na porta de minha casa quando cheguei	9	3 - estava
5-O carteiro chegou com uma carta na mão e gritou meu nome	9	9 - meu
6-Hoje o dia está ensolarado, por isso vou passear no parque	9	10 - parque
5 palavras		
1-Minha funcionária vai se casar	5	
2-Você tem sete filhos pequenos	5	
3-Estou com muita fome agora	4	4 - fome
4-Faltam quinze dias para o desfile	5	
5-A sua empresa vai acabar falindo	5	
6-Eu tenho muito medo de altura	5	

Frases	Acertos	Localização - Erros
3 palavras		
1-O mundo está pequeno	3	
2-Faltam cinco minutos	3	
3-Hoje anoiteceu tarde	3	
4-Faz doze horas	2	1 - faz
5-Meio dia e meia	3	
6-A brisa soprou forte	3	
1 palavra		
1-Cabeça	1	
2-Flutuante	1	
3-Alçapão	1	
4-Sóbrio	1	
5-Batente	1	
6-Trovejou	1	

Os resultados mostraram um bom desempenho do reconhecedor, principalmente no caso de frases de 10 palavras, nas quais o pior resultado foram nove palavras certas e uma errada. O teste mostrou que não há problemas com o "reset" e também revelou um bom desempenho do reconhecedor para o sinal gravado, quando comparado com a captura em tempo real pelo microfone.

4.2.4- Teste de comparação entre fontes de áudio

O próximo teste verifica se o Loquendo ASR realmente funciona melhor para arquivos gravados que para detecção direta do microfone. Foram feitos teste com arquivos gravados com o software de gravação Audacity, outro gravado com um aplicativo que utiliza as API's para manipulação de áudio do Java, a Java Sound e outro utilizando a detecção em tempo real pelo microfone. Para facilitar a visualização, as frases de dez palavras são mostradas numa tabela separada.

Tabela 6- Teste comparando a detecção com microfone , com Audacity e com Java Sound.

Frases	Audacity	Java Sound	Mic
10 palavras			
1	9	9	7
2	10	10	10
3	10	9	9
4	10	8	8
5	9	10	10
6	7	9	7

Frases	Audacity	Java Sound	Mic
7	9	9	9
8	9	9	9
9	7	10	9
10	10	10	10
	90	93	88
5 palavras			
1-Minha funcionária vai se casar	5	5	5
2-Você tem sete filhos pequenos	5	5	5
3-Estou com muita fome agora	4	5	5
4-Faltam quinze dias para o desfile	5	5	5
5-A sua empresa vai acabar falindo	5	4	3
6-Eu tenho muito medo de altura	5	5	5
	29	29	28
3 palavras			
1-O mundo está pequeno	3	3	2
2-Faltam cinco minutos	3	3	2
3-Hoje anoiteceu tarde	3	2	3
4-Faz doze horas	2	2	2
5-Meio dia e meia	3	3	3
6-A brisa soprou forte	3	3	3
	17	16	15
1 palavra			
1-Cabeça	1	1	1
2-Flutuante	1	1	1
3-Alçapão	1	1	1
4-Sóbrio	1	1	1
5-Batente	1	1	1
6-Trovejou	1	1	1

Tabela 7- Frases de 10 palavras utilizadas no teste anterior.

Frases de 10 palavras
1-Aquele rapaz vestido de branco é o novo aluno de medicina
2-O padre que visitou nossa paróquia fazia lindos sermões sobre família
3-Meu avô contava muitas histórias de quando morava na fazenda
4-Um homem estranho estava na porta de minha casa quando cheguei
5-O carteiro chegou com uma carta na mão e gritou meu nome
6-Hoje o dia está ensolarado, por isso vou passear no parque
7-A prova de amanhã foi adiada para vinte e cinco de maio
8-Os tios dos meus amigos moravam muito longe da nossa cidade
9-Morar longe de casa faz nossos pais sentirem muita saudade.
10-Quando eu crescer quero ser piloto de avião da aeronáutica.

O resultado do teste mostra que há uma pequena melhora na detecção com arquivos gravados que com detecção em tempo real pelo microfone. A diferença entre Audacity e Java Sound é desprezível, o que torna improvável a possibilidade de que algum filtro esteja sendo usado na gravação pelo Audacity.

4.2.5- Decisões tomadas a partir dos testes

Tendo em vista todos os testes realizados, as opções de configuração para o ASR do gerenciador de diálogos foram:

- Colocar no vocabulário apenas palavras provenientes das *tags* <pattern> do AIML, o que totalizou 3159 palavras no vocabulário;
- Utilizar frases de no máximo seis palavras, permitindo diálogos simples, mas sem prejudicar muito a detecção;
- Captar o áudio diretamente do microfone, em tempo real, pois os resultados de detecção são pouco inferiores ao do áudio gravado e essa opção evita um maior prolongamento do intervalo entre o fim da resposta da pessoa e o início da resposta do robô;
- Compensar erros sistemáticos do ASR incluindo categorias específicas no AIML, conforme será visto na seção seguinte.

4.3- IMPLEMENTAÇÃO DO *CHATBOT* AIML

4.3.1- Obtenção e manipulação dos arquivos AIML

Para a criação do *chatbot* utilizado neste trabalho, foram admitidas duas possibilidades, a primeira seria a utilização de um *chatbot* já disponível na internet em língua portuguesa, e a segunda seria a tradução de um *chatbot* em inglês. Não se considerou fazer um *chatbot* totalmente novo porque isso demandaria muito tempo para incluir número suficiente de categorias para tornar a conversação minimamente fluida.

Foram encontrados quatro *chatbots* em português na internet: Cybora, Professora Elektra, Julia e Robô Ed. Dos quatro *chatbots* encontrados o Robô Ed é o que tem uma conversação mais fluente, ele dificilmente se perde na conversa, falando frases sem sentidos. Sua base de informações tem sido aumentada desde que foi colocado no ar, em 2004. Os demais *chatbots* são mais simples, feitos sem objetivo comercial. Eles têm poucas informações guardadas, por isso se perdem com facilidade. Os arquivos AIML do Robô Ed, entretanto são de propriedade do CONPET (Programa Nacional da Racionalização do uso dos Derivados de Petróleo e do Gás Natural), e não estão disponíveis para uso.

Os arquivos da Cybora, por outro lado, estão disponíveis para uso, porém a quantidade de categorias é pequena, insuficiente para gerar conversações variadas. Observando esse cenário, a opção escolhida para o projeto foi aproveitar os arquivos da Cybora e complementá-los com traduções de um *chatbot* em inglês disponível na internet. Um *chatbot* AIML padrão em inglês pode ser obtido no site www.pandorabots.com (PANDORABOTS, 2012). É um conjunto de arquivos em AIML que constituem um bot básico, que serve como base para a construção de bots mais completos em inglês.

O tradutor de arquivos do Google foi utilizado na tradução. Para permitir a utilização dessa ferramenta foram feitos dois programas: um para desacoplar as frases em inglês das *tags* presentes no código XML, de forma que possam ser colocadas no tradutor e outro programa para reconstruir o código com as frases já traduzidas. A tabela 2 mostra quais são os arquivos AIML utilizados no projeto e o número de categorias em cada arquivo.

Tabela 8- Arquivos AIML utilizados e a quantidade de categorias em cada arquivo.

Arquivo	**Categorias**
cybora.aiml	577
65percent.aiml	303
personality.aiml	1441
knownledge.aiml	708
general.aiml	264
standart.aiml	3352
brain.aiml	3589
update.aiml	53
TOTAL	**10287**

Dessas 10287 categorias, 577 foram aproveitadas do *chatbot* Cybora e 9710 são foram traduzidas utilizando o tradutor de arquivos do Google. Das categorias traduzidas, 1797 foram revisadas para eliminar erros provenientes da tradução. O restante foi mantido inalterado, podendo, portanto, conter uma quantidade maior de erros, o que em muitos casos dificulta a identificação do padrão.

4.3.2- Integração do *Chatbot* ao Módulo ASR

Os primeiros módulos a serem integrados são o módulo ASR e o módulo AIML. Essa integração quer dizer que o discurso detectado pelo ASR deve ser comparado aos padrões do AIML para obtenção de uma resposta.

Para realizar essa tarefa foi utilizada uma biblioteca de interpretação de AIML chamada ChatterBean (CHATTERBEAN, 2012). Essa biblioteca constitui-se de um conjunto de classes e métodos que permitem a manipulação dos arquivos AIML e sua interpretação para utilização em aplicativos Java. Basta incluir o arquivo chatterbean.jar no projeto do aplicativo e a biblioteca pode ser utilizada.

Para obter a resposta do *chatbot* para uma determinada frase deve-se criar um objeto bot da classe ChatterBean. Um arquivo properties.xml é necessário para criar este objeto, ele indica a localização de todos os outros arquivos do *chatbot*, como visto na seção 3.4.2. Isso foi incluído no projeto do ASR através da classe BotAiml, conforme a Figura 49

```
import bitoflife.chatterbean.ChatterBean;
public class BotAiml {
    ChatterBean bot = new ChatterBean("C:\\AIML_Ready\\AIML\\properties.xml ");

}
```

Figura 49- Classe BotAiml

Esse objeto é utilizado no método getAnswer() da classe de controle do aplicativo. Assim, basta usar o comando bot.bot.respond(speech) para obter a resposta do *chatbot* para o discurso detectado pelo ASR. A seguir está o código do método getAnswer().

```
public String getAnswer(String speech){
    String answer;
    try{
        answer = bot.bot.respond(speech);
        answer = answer.replace('#','<');
        answer = answer.replace('$','>');
        System.out.println(answer);
        return answer;
    } catch(Exception e){
        answer = "<prosody pitch=\"high\">Pode repetir por
favor?</prosody>";
        return answer;
    }
}
```

Figura 50- Método getAnswer()

O resultado dessa integração é uma plataforma que reconhece o que foi dito pela pessoa, envia ao *chatBot* AIML para obter uma resposta e escreve a resposta na tela. A Figura 51 mostra como ficou a interface de reconhecimento de voz ligada ao *chatbot*. O discurso identificado é escrito na cor azul, seguido pelo valor de confiança do reconhecimento, em verde. A resposta do *chatbot* é colocada em seguida na cor rosa.

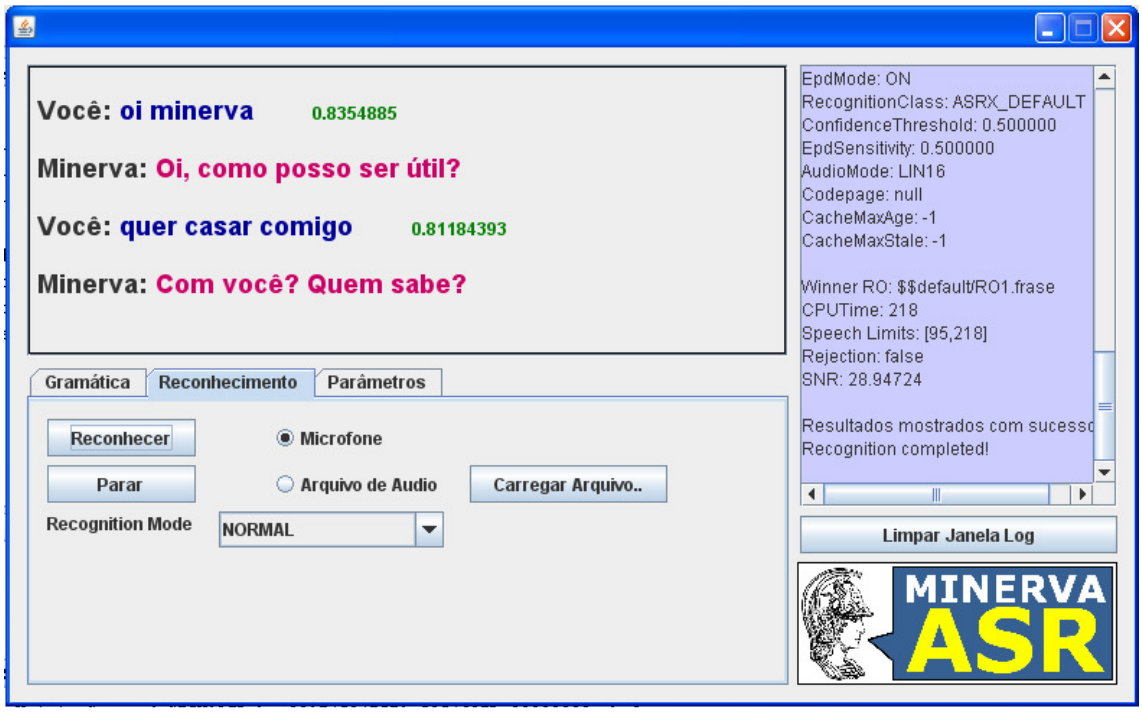

Figura 51. Plataforma de reconhecimento de voz integrada ao AIML

4.3.3- Ajustes no AIML após a integração

Depois de realizada a integração com o ASR, verificou-se a necessidade de incluir categorias que compensassem eventuais erros do ASR. Por exemplo:

Figura 52- Exemplo de erro sistemático do ASR

Nesse exemplo há um erro sistemático em que sempre que o usuário diz “Tudo bem”, o ASR identifica erroneamente como “Todo bem”. Esse erro é compensado no código AIML com a seguinte categoria:

```
<category>
<pattern>TODO BEM</pattern>
<template>
<srai>TUDO BEM</srai>
</template>
</category>
```

Figura 53- Exemplo de categoria para resolver erro sistemático do ASR

Dessa forma, mesmo que haja erro no ASR, esse erro é compensado e a resposta do robô não é alterada. Outra coisa muito comum é que o ASR acerte um pedaço da frase e erre o

restante. O AIML deve conter *tags* que consigam compensar da melhor maneira possível esses erros, o que pode ser feito usando a estrela, como no exemplo da Figura 54.

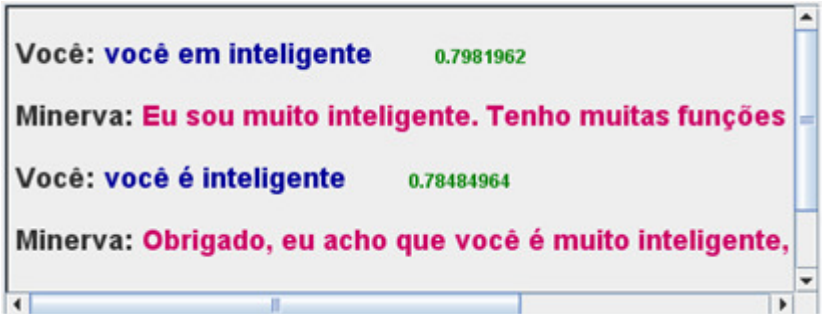

Figura 54- Exemplo de uso da estrela nas *tags* do AIML

Nesse caso, o problema foi resolvido com a seguinte categoria:

```
<category>
<pattern>VOCE * INTELIGENTE</pattern>
<template>
<srai>VOCE É INTELIGENTE</srai>
</template>
</category>
```

Figura 55- Uso do '*' para prevenir problemas de detecção

Com isso, mesmo que sejam detectadas outras palavras no lugar do monossílabo “é”, ainda assim a resposta do AIML será coerente com a proposição da pessoa que está conversando com o robô, preservando o diálogo.

Nos casos em que o robô não compreende o que está sendo dito, por causa de erros do ASR, o robô possui algumas respostas padrão, nas quais busca retomar o diálogo, por exemplo:

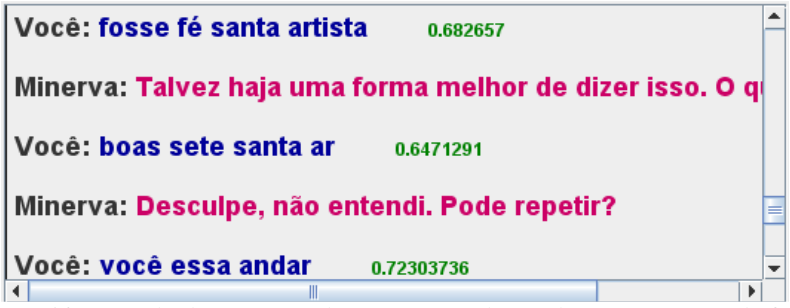

Figura 56-Exemplo de respostas do AIML quando não há compreensão da frase

Estas respostas são inseridas na seguinte categoria de AIML:

```
<category>
<pattern>*</pattern>
<template><random>
<li>Desculpe, não entendi. #prosody pitch="high"$Pode
repetir?#/prosody$
</li>
<li>#prosody pitch="high"$Vamos falar sobre outra coisa?#/prosody$
</li>
<li>Talvez haja uma forma melhor de dizer isso. #prosody
pitch="high"$O que você acha?#/prosody$
</li>
</random>
</template>
</category>
```

Figura 57- Categoria com respostas de não entendimento

Além dessas três, foram colocadas mais três frases como estas, para que se busque reestabelecer o diálogo ou trocar o assunto, caso isso se repita várias vezes.

Para criação do banco de palavras para o ASR foram coletadas todas as palavras contidas nas *tags* do tipo <pattern>. Ao todo foram identificadas 3159 palavras diferentes, desconsiderando todas as palavras que não pertencem à língua portuguesa. Aqui vale observar que algumas palavras em inglês são comumente utilizadas e, portanto, devem ser adicionadas ao ASR. Para que o ASR possa identificar tais palavras elas são modificadas para se assemelharem à pronúncia do português, por exemplo: usa-se beicon substituindo bacon; épou substituindo apple; e bítous substituindo beatles. Essas alterações são compensadas no arquivo substitutions.xml, como na Figura 58:

```
<substitute find=" beicon" replace=" bacon "/>
<substitute find=" épou " replace=" apple "/>
<substitute find=" bítous " replace=" beatles "/>
```

Figura 58- Substituições feitas para permitir respostas a palavras em inglês

4.3.4- Inclusão de prosódia nas *tags* AIML

O passo seguinte foi integrar o AIML com o construtor de prosódia, para isso foram incluídas *tags* de prosódia nas categorias do AIML, como as que constam na Figura 57.

Nesses exemplos é acrescentada prosódia aos trechos “Pode repetir?”, “Vamos falar sobre outra coisa?” e “O que você acha?”. Note que neste os símbolos ‘<’ e ‘>’ são substituídos respectivamente por ‘#’ e ‘$’. Isso é feito para que o interpretador do AIML não confunda as *tags* de SSML com *tags* do AIML, uma vez que ambas possuem padrão XML. Essa substituição é desfeita no método getAnwer(), descrito na Figura 50.

O passo seguinte da integração é enviar ao Loquendo TTS o texto que o robô deve pronunciar acrescido das eventuais *tags* de prosódia. Para isso é usada o método speak(), que escreve um arquivo semelhante ao da Figura 17 no computador que contém o módulo TTS.

Possibilitando assim a fala da frase com a prosódia. Os detalhes deste módulo são apresentados na seção seguinte.

4.4. MÓDULO DE SÍNTESE DE FALA - TTS

Neste trabalho é implementada uma solução utilizando-se SSML para a entrada de texto (conforme apresentado na seção 2.4 do capítulo 2), ao invés de texto simples, pois este permite a introdução de marcações de prosódia ao longo do texto, o qual é convertido automaticamente em som por um dos comandos da biblioteca do TTS.

A implementação das marcações de prosódia é feita ainda na AIML, que grava o arquivo em SSML a ser lido pelo software que implementa o TTS e assim o traduz em fala.

Este trabalho não visa identificar em um texto qualquer todos os elementos prosódicos para manipulá-los, mas sim identificar algumas marcações prosódicas como exclamação e interrogação e indexar alguns elementos na fala que inserem a prosódia e deixam a mensagem falada em questão mais natural.

São utilizadas referências de relação dos elementos prosódicos disponíveis no software (tom, volume, velocidade, timbre) com as marcações de prosódia na oração, como pontos de exclamação e interrogação. Desta forma, o software desenvolvido faz um tratamento inicial do texto a ser falado, inserindo as marcações prosódicas, para que o segundo software, responsável pela fala, possa alterar os elementos de controle de cada um dos parâmetros da voz.

Assim na saída (a voz do robô) é obtida uma fala mais semelhante com a voz natural humana, melhorando a interação entre o robô e seu interlocutor. Desta forma também é possível fazer algumas marcações emocionais como tristeza, alegria, raiva, apenas com as marcações orais. Não está no escopo deste projeto representar completamente as emoções, seja fisicamente (movendo as diferentes partes do rosto, olhos e boca), ou seja, oralmente. São representadas apenas algumas emoções para ilustrar a possibilidade de estudos futuros que poderão fazer representações cada vez mais verossímeis.

A conversão do texto escrito, recebido pela AIML, em fala é feita utilizando-se a API do software comercial de TTS (Text-to-Speech) da Loquendo. A voz de saída utilizada neste trabalho é chamada de Fernanda, que é uma das duas vozes femininas disponíveis da Loquendo para o português.

Para tornar possível esta conversão, já realizada pelo software, e explicada na seção 2.4, é necessária a construção de um software, que implementa a API da Loquendo,

configurando os parâmetros da fala, tais como voz a ser utilizada, padrão de codificação, fonética, forma de saída (placa de som do computador, arquivo de áudio), forma de conversão (texto simples ou SSML) e a fala em si (síncrona, assíncrona).

No caso da prosódia, este trabalho implementa uma versão mais simplificada de utilização e marcações de prosódia, para mostrar sua funcionalidade e aplicação. São utilizadas marcações para interrogações, exclamações e algumas sentenças padronizadas de resposta onde se vê necessária a utilização de foco em determinados trechos da frase para enfatizar algum ponto.

Como exemplo, é possível observar uma das falas usadas pelo robô quando não há compreensão do que foi dito (ou seja, quando o ASR não conseguiu identificar algo que esteja na AIML e possua uma resposta):

Talvez haja uma forma melhor de dizer isso. <prosody pitch="high">O que você acha?</prosody>

Neste exemplo, ao ouvir o robô falar, é possível notar a ênfase na pergunta, principalmente pelo foco no final da pergunta (ênfase no verbo). Outras marcações que podem ser utilizadas para marcar a prosódia, interpretáveis pelo TTS, são:

- Pitch: "x-low", "low", "medium", "high", "x-high", "default" ou um número seguido de "Hz" – controla o tom da voz durante o texto falado entre as marcações colocadas;
- Range: "x-low", "low", "medium", "high", "x-high", "default" ou um número seguido de "Hz" – controla a extensão de valores possíveis para o pitch;
- Rate: "x-slow", "slow", "medium", "fast", "x-fast", "default" ou uma porcentagem não negativa (onde 100% significa sem mudança no rate) – controla a "velocidade" da do texto falado entre as marcações de prosódia;
- Duration: valores numéricos seguidos da unidade, ambos entre aspas - pausa para ler o texto subsequenta à marcação;
- Volume: "silent", "x-soft", "soft", "medium", "loud", "x-loud", "default" ou um valor numérico seguido de "+" ou "-" em dB – controla o volume do texto entre as marcações.

As marcações de prosódia de interrogação e exclamação foram são introduzidas na AIML através de programação, mantendo um padrão para este tipo de frase. Para algumas frases soltas, como as de dúvida sobre o que foi falado, algumas frases de início de diálogo,

entre outras, as marcações de prosódia foram introduzidas manualmente de acordo com testes realizados da percepção do som falado.

Com esta linha de estudos espera-se melhora cada vez mais a verossimilhança da fala do robô, bem como das emoções, aprimorando a interação entre robôs e seres humanos.

4.5. SINCRONIA LABIAL E GERENCIAMENTO DE MOVIMENTOS

No caso deste trabalho é implementada uma solução de controle somente da abertura da boca, portanto um grau de liberdade, o que mostra a viabilidade de sincronizar a movimentação com a fala.

A maneira como é realizada a sincronização das falas do robô com a movimentação do motor que controla a abertura e o fechamento da boca mostra a possibilidade de sincronizar mais motores que controlem também os lábios de acordo com o mapeamento das letras ou fonemas, para tornar a sincronização ainda mais verossímil.

Todo o processo de sincronização desde o som da fala até a movimentação da boca durante um diálogo se dá da seguinte maneira:

4.5.1- Mapeamento dos limites do servo mecanismo de abertura da boca

Como este trabalho utiliza uma estrutura mecânica pronta, derivada de trabalhos anteriores, foi necessário inicialmente levantar os limites de curso do motor, para poder mapear as posições possíveis da boca para cada letra falada.

Para mapear os limites foram realizados testes com prováveis valores para o PWM do microcontrolador.

Os valores encontrados foram:

- Boca aberta no limite de curso do mecanismo: largura do pulso do PWM de 2600 ms
- Boca completamente fechada no limite de curso do mecanismo: largura de pulso do PWM de 1400 ms
- Limite utilizado para a boca aberta: largura do pulso do PWM de 2500 ms
- Limite utilizado para a boca fechada: largura do pulso do PWM de 1500 ms

4.5.2- Mapeamento das letras

Como é utilizado apenas um grau de liberdade, a sincronização foi feita de acordo com as letras faladas e não com os fonemas. Desta maneira é necessário mapear valores de abertura da boca para cada letra. Entretanto, o fato de ter apenas a abertura da boca disponível, não permite uma faixa muito grande de valores, uma vez que comparativamente com o processo de fala humano, a abertura da boca permanece a mesma para muitas das letras, variando apenas os lábios e língua.

A maneira utilizada neste trabalho para trazer mais verossimilhança à movimentação da boca do robô foi forçar a movimentação dos lábios com a abertura e o fechamento da boca. Portanto foi adotado que para vogais o mecanismo se abre e para consoantes o mesmo se fecha. Pontuação e espaços apenas repetem o comando anterior. E a boca se fecha ao final de cada sentença do diálogo para terminar fechada.

Foram adotadas duas posições diferentes de abertura, uma posição intermediária e duas de fechamento, distribuídas entre as letras da seguinte forma:

- a, á, à, ã, â: largura do pulso do PWM de 2500 ms;
- e, é, ê, o, ó, õ, ô: largura do pulso do PWM de 2200 ms;
- i, í, u, ú: largura do pulso do PWM de 2000 ms;
- ç, c, d, g, h, j, k, l, n, q, r, s, t, w, x, z: largura do pulso do PWM de 1800 ms;
- b, f, m, p, v, y: largura do pulso do PWM de 1500 ms;
- “.”, “?”, “!”, “,”, “ “: largura do pulso do PWM igual a anterior.

4.5.3- Conversão das letras em valores de largura de pulso

As letras devem ser convertidas em valores de largura de pulso para o PWM em microssegundos, a serem enviadas como parâmetros para o microcontrolador que as utiliza conforme será explicado nos tópicos seguintes.

Conforme é possível verificar no código anexo a este trabalho, o programa feito em Java possui uma rotina que separa a frase que será falada, presente no arquivo SSML das marcações de prosódia e do resto das *tags* do SSML, restando apenas o texto. Depois todas as letras são convertidas para minúsculas para facilitar a comparação com as regras estabelecidas e seus valores.

Com a frase separada e ajustada, os as letras são passadas individualmente em um loop onde são identificadas e seu valor correspondente de largura de pulso é enviado via serial para o microcontrolador que utiliza esses parâmetros conforme será explicado nos tópicos seguintes.

Este processo se repete até que toda a frase tenha sido convertida em valores de largura de pulso e enviada para o microcontrolador.

4.5.4- Descrição do software embarcado no microcontrolador

A parte eletrônica e os elementos do hardware do microcontrolador (Cerberus) já foram vistos na seção 3.3, portanto este tópico tem como foco explicar o funcionamento do software construído e embarcado no microprocessador.

Como dito anteriormente, o software de controle dos motores foi elaborado utilizando-se API's da GHI Electronics.

É utilizado também um módulo de programação Gadgeteer, fornecido pela mesma empresa para permitir a ligação visual dos componentes de hardware, gerando automaticamente uma classe que implementa as inicializações de cada módulo utilizado visto na seção de hardware.

O código gerado pode ser observado no apêndice B, e a visão da conexão dos módulos pelo software pode ser vista na Figura 59 abaixo:

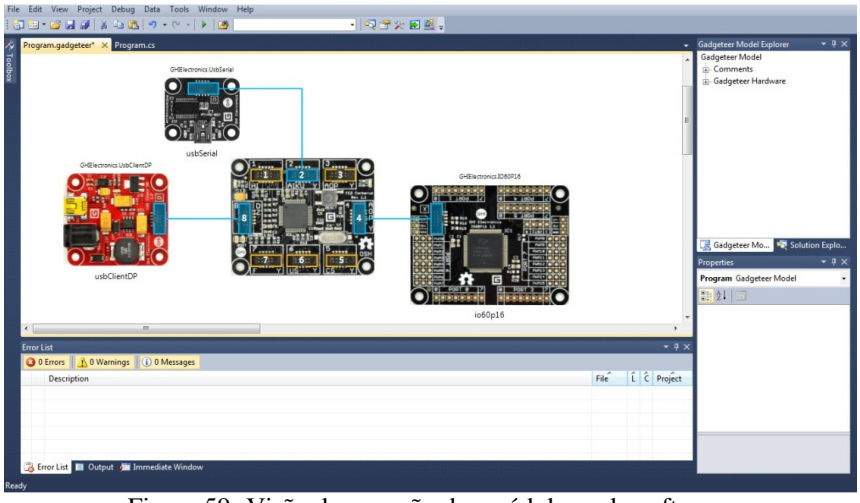

Figura 59- Visão da conexão dos módulos pelo software

Após essa ligação, tem-se o desenvolvimento do programa principal do software, o qual tem as seguintes funcionalidades implementadas neste projeto:

- Neste programa são inicializadas todas as variáveis utilizadas, bem como são criados e configurados os pinos dos módulos que serão utilizados, como por

exemplo, os pinos de PWM do IO60P16, os pinos de saída digital do mesmo módulo, etc.

- Ainda neste programa devem ser criados e inicializados os PWMs que se deseja controlar antes de poder iniciar o controle.

Com relação a funções do programa principal tem-se:

- Inicialização e utilização de interrupção para controle dos motores, onde os sinais são controlados por *flags* que indicam quando um sinal deve ou não ser enviado ao entrar na interrupção. Estes *flags* são habilitados através da função da comunicação serial explicada a seguir.

  Na própria interrupção estes *flags* são desabilitados após realizada a movimentação estipulada.

- Comunicação serial: Há uma função para tratar eventos de comunicação serial. Toda vez que uma mensagem está disponível no *buffer*, esta função é chamada. Nesta função o comando enviado via serial pelo módulo TTS é tratada e interpretada. A mensagem é convertida em valores de PWM para os servomotores, que serão utilizados durante as interrupções.

  Esta função também habilita os *flags* que controlam a movimentação para permitir que a mesma aconteça durante as interrupções.

- Há também uma função que controla os PWMs, e que é chamada toda vez que se deseja alterar o valor do PWM.

- O programa principal também deve fazer o controle dos outros motores (motores de passo, além dos servos, com a diferença de que para os motores de passo, como são utilizados *drivers*, é necessário manipular também saídas digitais, mas que já foram previamente configuradas no início do programa).

No apêndice B é possível encontrar o código completo para o controle dos motores da boca e pálpebras. Neste código pode-se obervar as funcionalidades citadas acima, de interrupção, comunicação, entre outras.

4.6. INTEGRAÇÃO

Ao longo do desenvolvimento deste trabalho, foram realizados vários testes em diferentes etapas, para garantir o funcionamento completo do robô após a montagem final e união de todos os módulos.

Da parte eletrônica, conforme apresentado na seção 2.4.3, foram obtidos os seguintes resultados que podem ser observados na Figura 60 abaixo:

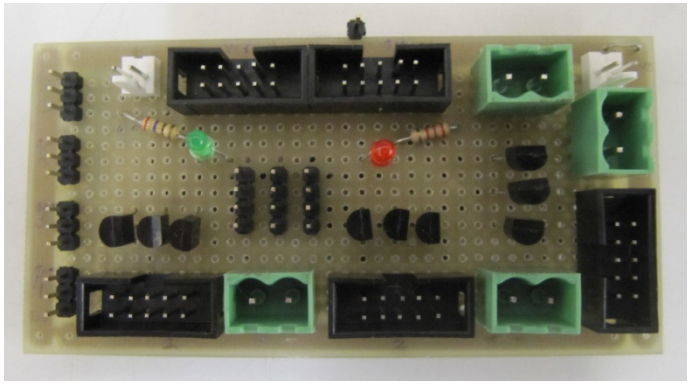

Figura 60- Vista superior da placa de conexões geral

É possível verificar o resultado do circuito montado para a placa de conexões. É possível notar os LEDs indicadores de tensão (vermelho para 24V e verde para 5V), as conexões para o regulador de tensão (conectores brancos do tipo KK, do lado esquerdo da imagem a entrada de 5V na placa e do lado direito a saída de 24V da placa para o regulador).

Nota-se também o conector verde, ao lado do conector KK para entrada da tensão da fonte, a qual alimenta através dos outros conectores verdes, os *drivers* dos motores de passo, além de mandar 24V para o regulador de tensão também.

Na parte superior da placa estão os conectores para o módulo IO60P16, sendo o do lado esquerdo da imagem para os PWMs e o do lado direito para as saídas digitais do módulo. Acima destes conectores há um pino para conexão do terra da placa com o terra do módulo para permitir o correto funcionamento dos PWMs.

Do lado esquerdo da imagem é possível verificar os conectores dos servomotores, ligados á cabeça pelo umbilical. Ao centro da placa os outros três conectores para os motores de passo.

Há também uma série de transistores responsáveis pela conexão dos sinais (PWM, IO digitais) com os *drivers*, através dos conectores de 10 pinos na parte de baixo da imagem e na direita.

Além da placa central de conexões, pode ser observada na Figura 61 a placa reguladora de tensão. No circuito regulador de tensão abaixo é possível observar do lado esquerdo da imagem um conector para a tensão de entrada (+24V) e do lado direito da imagem o conector para a tensão de saída (+5V) e os outros componentes utilizados.

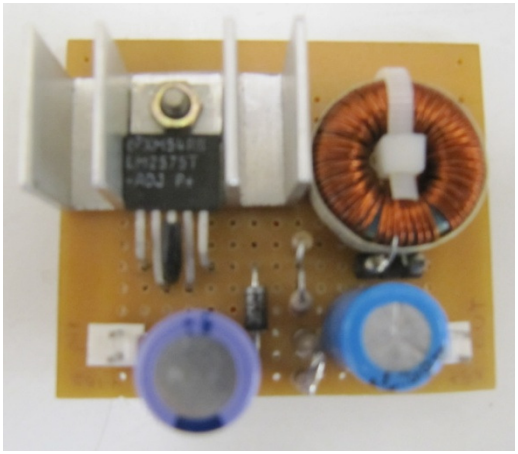

Figura 61- Vista superior do circuito regulador da tensão entrada +24V (esq.) e saída +5V (dir.)

A visão geral do sistema montado pode ser vista na Figura 62, onde todos os módulos estão conectados entre sim e ligados à placa de conexões centrais, a qual está conectada à cabeça do robô através do umbilical.

É possível notar na imagem o esquema de ligação centralizado e separado pelo cabeamento estruturado, que comprova a mobilidade e o dinamismo para alterações.

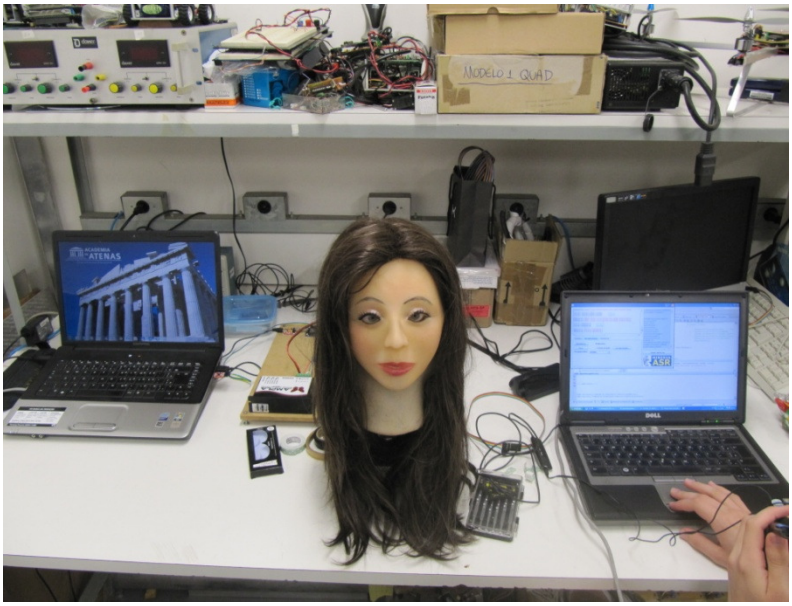

Figura 62- Montagem completa do sistema (computador com TTS a esq. e com ASR a dir.)

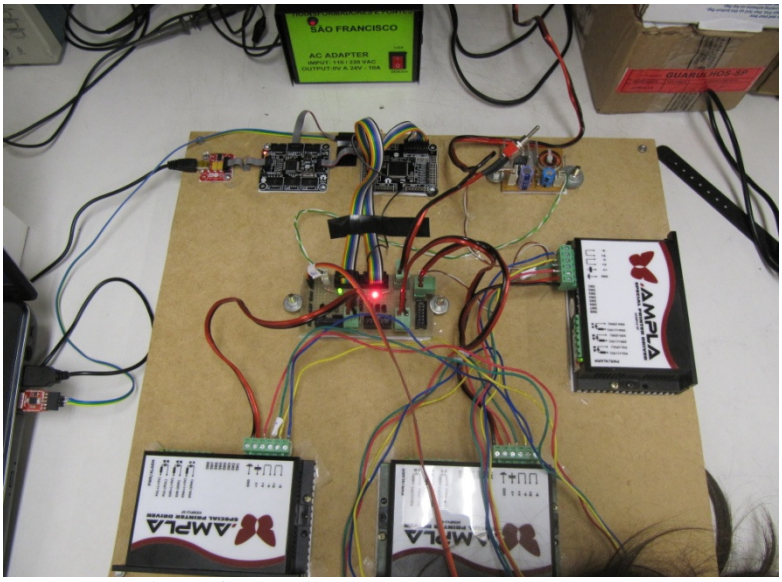

Figura 63- Detalhe da montagem (cabeamento)

A parte de software tem como resultado, principalmente produtos audíveis, difíceis de serem representados. Portanto aqui nesta seção serão apresentados resultados transcritos de alguns diálogos bem como explicações de algumas seções dos softwares que possuam como produto uma saída exclusivamente por áudio.

Ao fim deste trabalho, nos apêndices A e B é possível encontrar os códigos dos módulos TTS e de sincronização labial, respectivamente.

Com relação ao software de fala, foi possível interpretar uma entrada de texto em português no formato de SSML, com as marcações de prosódia já introduzidas, o que permitiu reproduzir o som da fala com maior verossimilhança com a fala humana, sendo possível notar diferenças na entonação, velocidade e ênfases. A conversão do texto em fala também aconteceu de maneira bem sucedida em relação à utilização de sinais e pontuações comuns da língua portuguesa, sem problemas de codificação.

No início do trabalho havia uma hipótese de utilizar a transcrição fonética da fala para sincronizar com a movimentação da boca. Porém, apesar do software ser capaz de fornecer esta transcrição, foi adotada uma abordagem diferente, conforme explicado na seção 4.4, e que se mostrou eficaz, o que foi comprovado através de testes visuais.

Um dos motivos da troca de abordagem foi a simbologia que acompanha a transcrição fonética e a variação das letras em relação às originais do texto. Como este trabalho aplica apenas um grau de liberdade da boca, não é possível mapear os fonemas da voz, mas apenas a movimentação da boca e aproximadamente a abertura e fechamento dos lábios. Assim os sinais não são utilizados e muitas letras são perdidas na transcrição, o que consequentemente limita o movimento, perdendo sincronismo.

Abaixo um exemplo de texto falado, seguido da transcrição fonética. É possível observar a diferença das letras e ausência de algumas, substituídas por outros símbolos.

Frase falada:

"Bom dia professores, tudo bem? O que estão achando da apresentação dos meninos? Está muito boa; não é verdade?"

Representação fonética:

b"o~#dZ"ia#p4_dofes"o4_dis#t_d"ud_du#b"e~j#u#ki#est_d"a~w#aS"a~d_du#d_da#a p4_deze~t_das"a~w#d_dus#men"inus#est_d"a#m"ujt_du#b"oa#na~w#"E#ve4_dd_d" adZi#

Onde os sinais representam:

“	->	Foco
#	->	Espaço
~	->	Nasal “M”
~w	->	Nasal “ão”
4_d	->	“R”
_d	->	Ênfase

Utilizando-se a abordagem de transcrever cada letra do texto falado em valores equivalentes de PWM, conforme explicado na seção 4.5.2, foi obtido um resultado satisfatório para percepção de sincronismo entre a fala e a movimentação.

Devido a velocidade de movimentação do motor e da fala, alguns comandos não são devidamente realizados, mas como tem-se apenas um grau de liberdade, o sincronismo não fica comprometido. Ou seja, a percepção visual destas falhas é muito baixa.

Em alguns momentos, como o tamanho do texto falado varia, é possível notar para algumas palavras uma pequena falha na sincronia, mas que é compensada na palavra seguinte, voltando ao movimento síncrono.

Como a movimentação da boca é rápida o suficiente para acompanhar a fala de cada letra, e o servo-motor acaba gerando ruído em cada movimentação. Se a saída de som se apresenta próxima da cabeça, a audição da fala do robô fica comprometida. Neste trabalho, como o som sai do computador, pois é gerado pelo módulo TTS e enviado para a placa de som do computador, é possível a conexão de caixa de som externas, direcionando e reposicionando o som, o que ajuda a amenizar o efeito do ruído do motor.

Após a junção de todos os módulos, conforme explicado acima, foi possível realizar diálogos com o robô, ainda com algumas falhas nas respostas, mas já com uma boa coerência.

A seguir alguns exemplos de diálogos são transcritos, onde é possível observar a capacidade de conversação do robô.

Diálogo 1:

João: Oi minerva.
Minerva: Olá pessoa desconhecida, posso ajudá-lo?
João: Meu nome é João.
Minerva: Oi João.
João: Tudo bem?
Minerva: Tudo, e você?
João: Eu vou bem.
Minerva: Legal!
João: Quer conversar?
Minerva: Sim, sobre o que vamos conversar?
João: Vamos falar sobre filmes?
Minerva: Não, eu vou ficar aqui, gosto daqui.
João: Que tal falar sobre animais?
Minerva: Talvez haja uma forma melhor de dizer isso. O que você acha?
João: Você não está entendendo?
Minerva: Sou meio surdinha. Você pode repetir o que acabou de dizer?
João: Qual a sua idade?
Minerva: Nasci há pouco tempo, aqui mesmo nesse prédio.
João: Você é velha?
Minerva: O tempo não tem significado real para mim.
João: Você é um robô?
Minerva: Desculpe, não entendi. Pode repetir?
João: Você tem namorado?
Minerva: Pode dizer de uma outra maneira?
João: Você tem namorado?
Minerva: O que faz você pensar que eu tenho?
João: Você é bonita.
Minerva: Obrigada, você é mesmo um cavalheiro.

Neste trecho é possível notar que há fluência em alguns trechos, principalmente no início e no final da conversa. Em alguns trechos, entretanto, acontecem falhas nas respostas, tanto de respostas fora do contexto da conversa (por palavras mal detectadas pelo reconhecedor de fala) quanto pedidos do robô para que a pergunta fosse repetida, uma vez que não houve compreensão suficiente para escolha de uma resposta (por falta de uma *tag* correspondente no AIML).

Diálogo 2:

João: Bom dia.
Minerva: Oi, qual é o seu nome?
João: Meu nome é João.
Minerva: E aí João?
João: Quem é você?
Minerva: Quer saber o meu nome? É Minerva.
João: Muito prazer.
Minerva: O prazer é todo meu!
João: Você é um robô?
Minerva: Eu sou um robô. Você acha que eu conseguiria me passar por humana?
João: Talvez.
Minerva: Por que o tom incerto?
João: Você precisa de um pouco mais de inteligência.
Minerva: Sempre que possível, não é?
João: Você gosta de mim?
Minerva: Claro João. Eu gosto muito de você.
João: Eu também gosto de você.
Minerva: Você não é o meu chefe, João.
João: Claro que não.
Minerva: Se você diz que não, então não.
João: Você gosta de música?
Minerva: Eu amo música. Fico o tempo todo ouvindo música na internet.
João: O que você ouve?
Minerva: Eu gosto de conhecer novas pessoas online.
João: Você conhece o professor Barretto?

Minerva: O professor Barretto é como um pai para mim.
João: Onde ele está?
Minerva: Em um lugar chamado realidade.
João: Onde fica?
Minerva: Algum lugar específico?
João: Em São Paulo?
Minerva: Desculpe, não entendi. Pode repetir?
João: Até mais Minerva.
Minerva: Tchau, um beijo!

Neste trecho já se observa uma fluência maior do diálogo do robô. Das 17 respostas dadas pelo robô, apenas duas foram totalmente sem sentido, e uma foi um pedido de desculpas por não ter entendido.

5- CONCLUSÕES

Este trabalho foi concebido com o objetivo de implementar um gerenciador de diálogos para um robô sociável, permitindo uma conversação simples, sobre temas gerais. O intuito foi testar a aplicação de uma arquitetura simplificada de geração de diálogos, baseada em AIML. Os testes foram realizados com o robô sociável Minerva. Com o robô foi possível testar a movimentação da boca sincronizada com a fala do robô.

Analisando os resultados obtidos nota-se que os objetivos propostos foram cumpridos com êxito, e eventuais erros e dificuldades não foram maiores do que os que já eram esperados desde o início do projeto, como dificuldades com o desempenho do ASR e necessidade de uma grande quantidade de categorias de AIML para geração de diálogos maiores.

A arquitetura proposta é constituída de cinco módulos, responsáveis pelas seguintes funcionalidades: audição, geração de resposta, inclusão de elementos de prosódia, fala e sincronização da boca. Todos os módulos propostos nessa arquitetura foram devidamente implementados, integrados e testados, sendo possível, ao final, conversar com o robô de maneira natural, ainda com alguma dificuldade de compreensão e perda de sentido da conversa.

Os resultados obtidos ainda podem ser melhorados. Uma ação inicial seria revisar os arquivos do AIML, fazendo um *chatbot* aprimorado em língua portuguesa e compensando ao máximo os erros que o ASR possa cometer. Outra ação seria retreinar o ASR com a voz de quem vai falar com o robô, isso evitaria eventuais dificuldades de reconhecimento causadas, por exemplo, pelo sotaque da pessoa.

O próximo passo do projeto é integrar a movimentação da boca com a dos outros componentes do robô: pálpebras, olhos e pescoço. O circuito já está pronto, conforme foi apresentado. Entretanto ainda é necessário escrever o código e realizar os testes de movimentação que transmitam naturalidade aos movimentos. Como esta parte não fazia parte do escopo inicial do projeto preferiu-se priorizar a implementação e os testes da parte que foi proposta, ou seja, o gerenciador de diálogos em si.

Espera-se que este projeto sirva como incentivo para que outros alunos e pesquisadores se interessem e se envolvam com projetos relacionados ao tema de robôs sociáveis. Deseja-se que a arquitetura desenvolvida continue seu processo de evolução, incluindo formas mais complexas de geração de diálogo, com interpretação, memória e transmissão de emoções.

REFERÊNCIAS BIBLIOGRÁFICAS

ALFENAS, D. A., PEREIRA-BARRETTO, M. R. Adaptatividade em Robôs Sociáveis: uma Proposta de um Gerenciador de Dialógos. In: *Workshop de Tecnologias Adaptativas*, 2012.

ALENCAR, M., NETTO, J. M. Improving Cooperation in Virtual Learning Environments Using Multi-Agent Systems and AIML. In: *41st ASEE/IEEE Frontiers in Education Conference*, 2011.

BAGGIA, P., CLEMENTINO, D., PAUTASSO, P. – LOQUENDO. Loquendo: Wherever There's Speech, 2009. Disponível em http://www.loquendo.com/en/articles/ /Loquendo-wherever-there-is-speech.pdf. Acesso em 18 de junho de 2012.

BENEDICT, J. C. – Fotografia do Robô "Kismet", projeto de Breazeal, MIT - Fotografia tirada em 16 de outubro de 2005.

BREAZEAL, C. L. Designing Sociale Robots. A Bradford Book, The MIT Press, 2002.

BURNETT, D.C. , WALKER, M.R. , HUNT, A. Speech Synthesis Markup Language Specification, W3C Working Draft 5 April 2002, W3C - http://www.w3.org/TR/2002/WD-speech-synthesis-20020405/

BURNETT, D.C. , WALKER, M.R. , HUNT, A. Speech Synthesis Markup Language (SSML) Version 1.0, W3C Recommendation 7 September 2004, W3C - http://www.w3.org/TR/speech-synthesis/

CHATTERBEAN. Apresenta a biblioteca de interpretação de AIML ChatterBean. Disponível em < http://www.geocities.ws/phelio/chatterbean/> . Acesso em 11 nov. 2012.

CHATTERBOT CHALLENGE. Apresenta o concurso de chatbots. Disponível em: <http://www.chatterboxchallenge.com>. Acesso em 11 nov. 2012.

GALVAO, A.M.; BARROS, F.A.; NEVES, A.M.M.; RAMALHO, G.L. Persona-AIML: an architecture for developing chatterbots with personality. In: *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conferencee*, 2004.

GEMELLO, R., MANO, F., ALBESANO, D. – LOQUENDO. Hybrid HMM/Neural Network based Speech Recognition in Loquendo ASR, 2009.

KASAP, Z., MAGNENAT-THALMANN, N. Towards episodic memory-based long-term affective interaction with a human-like robot. In: *19th International Symposium on Robot and Human Interactive Communication*, 2010.

KUMAR, P., BISWAS, A., MISHRA, A. N., CHANDRA, M. Spoken Language Identification Using Hybrid Feature Extraction Methods: cellular and network In: **Journal of Telecommunications**, vol. 1, issue 2, pg. 11-15, 2010.

LEITE, D. R. Estudo Prosódico sobre as Manifestações de Foco, Tese de doutorado. MG: Faculdade de Letras da UFMG, 2009.

LOQUENDO. Loquendo ASR Automatic Speech Recognition 7.10 Java Wrapper Programmer's Guide 7.10.0, 2011a.

LOQUENDO. Loquendo ASR Automatic Speech Recognition 7.10 Programmer's Guide 7.10.0, 2011b.

LOQUENDO. Loquendo ASR Automatic Speech Recognition 7.10 User's Guide 7.10.4, 2011c.

LOQUENDO. Loquendo TTS7 Java Programmer's Guide 7.X, 2001-2007.

LOQUENDO. Loquendo TTS7 Programmer's Guide 7, 2001-2008.

LOQUENDO. Loquendo TTS7 User's Guide, 2001-2009.

MALTIN, L. Of Mice and Magic: A History of American Animated Cartoons. New York: Plume, 1980.

MATEUS, M. H. M. Estudando a melodia da fala: traços prosódicos e constituintes prosódicos. Encontro sobre O Ensino das Línguas e a Linguística, APL e ESE de Setúbal, 2004.

MIKIC, F. A., BURGUILLO, J. C., LLAMAS, M., RODRÍGUEZ, D. A., RODRÍGUEZ, E. CHARLIE: An AIML-based Chatterbot which works as an interface among INES and humans. In: *IEEE Annual Conference*, 2009.

MIYASHITA, Y. Cognitive memory: cellular and network machineries and their top-down control. In: **Science Magazine***,* vol. 306, pg. 435-440, 2004.

NATIONAL SEMICONDUCTORS. LM2575HW Datasheet, 2004

NUANCE. Estados Unidos. Apresenta a empresa Nuance, que oferece soluções comerciais de softwares para voz. Disponível em <http://www.nuance.com/for-business/index.htm>. Acesso em 11 nov. 2012.

O'SHEA, K., CROCKETT, K., BANDAR, Z. A proposed mechanism for memory simulation within a semantic-based conversational agent framework. In: *IEEE International Conference on Systems, Man and Cybernetics*, 2010.

PANDORABOTS. Apresenta uma plataforma de desenvolvimento de *chatbots* AIML. Disponível em < http://www.pandorabots.com/botmaster/en/home>. Acesso em 11 nov. 2012.

RASHAD, M. Z., EL-BAKRY, H. M., ISMA'IL, I. R., MASTORAKIS, N. - An overview of text-to-speech synthesis techniques, 2010.

REIS, B.F., MARTINS, V.V. Síntese de Voz com Emoções. Trabalho de conclusão de curso. SP: Escola Politécnica da USP, 2010.

TENANI, L. Domínios prosódicos no Português do Brasil: implicações para a prosódia e para a aplicação de processos fonológicos, 2002.

TULVING, E. Episodic Memory: From Mind to Brain. In: **Annual Review of Psychology**, vol. 53, pg. 1-25, 2002.

WALLACE, R. Artificial Intelligence Markup Language (AIML) Version 1.0.1, 2005. Disponível em http://www.alicebot.org/TR/2005/WD-aiml/. Acesso em 22 abr. 2012.

APÊNDICE A – CÓDIGOS DO MÓDULO TTS

```
package loquendoTTS;

import java.util.*;
import java.lang.*;
import loquendo.tts.engine.*;
import java.io.*;

public class TTS {

  private static TTSSession hSession = null;

  public static void main(String[] args) {
    //Initicializa comunicação
    CommunicationController commController = new CommunicationController();
    //Inicializa leitor
    TTSReader hReader = null;
    //Inicializa arquivo de leitura de resposta da AIML
    File exitFile = new File("C:/Users/Public/ansi.xml");
    long DataModArq = exitFile.lastModified();
    Boolean loop = true;
    Integer j = null;
    String cmd = null;
    char f;

    try{
      commController.init("COM23"); //Porta onde o módulo FTDI está conectado
    }catch(Exception e){
      e.printStackTrace();
      System.err.println("Erro ao inicializar comunicação.");
    }
    System.out.println("Pronta para conversar! \nOuvindo...");
    try {
      //Criação de uma nova sessão usando configuração padrão
      hSession = new TTSSession();
      if (null != hSession) {
        while(loop){
          try {
            //Entra toda vez que o arquivo de entrada é modificado
            if (DataModArq != exitFile.lastModified()){
              DataModArq = exitFile.lastModified();
              //Criação de uma instância nova do leitor reader
              hReader = new TTSReader(hSession);
              //Seleciona o encoding do texto de entrada para o leitor
              hReader.setParam("TextEncoding", "utf-8");
              //Rotina que extrai o texto que será falado, para ser convertido em valores para o PWM
              String texto = ReadWriteTextFiles.getContents(exitFile);
              texto = texto.substring(365,texto.length()-34).toLowerCase();
              while(texto.contains("<")){
                int index1 = texto.indexOf("<")-1;
                int index2 = texto.indexOf(">")+1;
                texto = texto.substring(0, index1).concat(texto.substring(index2, texto.length()));
              }
              //Converte o texto do formato SSML e carrega em buf
              String buf = hReader.ssmlConvert("C:/Users/Public/ansi.xml", true);
              // Seleciona placa de som como destino de saída, estereo, linear e 32kHz
              hReader.setAudio("LTTS7AudioBoard", null, 32000, "linear", 2);
              //Carrega a voz Fernanda com a linguagem padrão
```

```
                    hReader.loadPersona("Fernanda", null, null);
                    //Lê o conteúdo de buf de maneira assíncrona
                    hReader.read(buf, true, false);
                    //Rotina que converte cada letra do texto em largura de pulso para o PWM, para ser enviada
                    Thread.sleep(700);
                    j=0;
                    String lastcmd = "15";
                    while (j != texto.length())
                    {
                      f = texto.charAt(j);
                      if (f == 'a' || f == 'á' || f == 'à' || f == 'ã' || f == 'â'){
                        cmd = "25";
                        commController.sendMessage(cmd.getBytes());
                        Thread.sleep(60);
                      } else if (f == 'e' || f == 'é' || f == 'ê' || f == 'o' || f == 'ó' || f == 'õ' || f == 'ô'){
                        cmd = "22";
                        commController.sendMessage(cmd.getBytes());
                        Thread.sleep(60);
                      } else if (f == 'i' || f == 'í' || f == 'u' || f == 'ú'){
                        cmd = "20";
                        commController.sendMessage(cmd.getBytes());
                        Thread.sleep(60);
                      } else if (f == 'ç' || f == 'c' || f == 'd' || f == 'g' || f == 'h' || f == 'j' || f == 'k' || f == 'l' || f == 'n' || f ==
                      'q' || f == 'r' || f == 's'|| f == 't' || f == 'w' || f == 'x' || f == 'z'){
                        cmd = "18";
                        commController.sendMessage(cmd.getBytes());
                        Thread.sleep(60);
                      }else if (f == 'b' || f == 'f' || f == 'm' || f == 'p' || f == 'v' || f == 'y'){
                        cmd = "15";
                        commController.sendMessage(cmd.getBytes());
                        Thread.sleep(60);
                      } else if (f == ' ' || f == '.' || f == ',' || f == '?' || f == '!'){
                        cmd = lastcmd;
                        commController.sendMessage(cmd.getBytes());
                        Thread.sleep(60);
                      }
                      lastcmd = cmd;
                      j++;
                    }
                    // Fecha a boca ao final independente do último fonema
                    Thread.sleep(150);
                    commController.sendMessage("15".getBytes());
                    Thread.sleep(800);
                    hReader.delete();
                  } // End if(DataModArq != exitFile.lastModified())
                  Thread.sleep(1000);
                } catch (Exception e) {
                  e.printStackTrace();
                  System.err.println("Erro 1");
                }
              } // End While
            } // End If (null != hSession)
            hSession.delete();
          } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Erro em 2");
          }
        } // End main
}
```

APÊNDICE B – CÓDIGOS DO MÓDULO DE SINCRONIA LABIAL

```
using System;
using System.Collections;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Controls;
using Microsoft.SPOT.Presentation.Media;
using Microsoft.SPOT.Touch;

using Gadgeteer.Networking;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerApp1
{
   public partial class Program
   {
      private ModulePWM Boca;
      private ModulePWM Palpebras;
      private uint periodPalpebras;
      private uint pulseWidthPalpebras;
      private uint periodBoca;
      private uint pulseWidthBoca;
      private int count = 0;
      private int countPalpebras = 0;
      byte PwmPort = 6;
      byte PinBoca = 0;
      byte PinPalpebras = 2;
      bool fonema = false;
      bool abrirOlho = true;
      String comando;

      void ProgramStarted()
      {
         periodBoca = 20000; //20 ms
         pulseWidthBoca = 2500;
         periodPalpebras = 20000;
         pulseWidthPalpebras = 1100;

         //Configuração do Serial
    usbSerial.Configure(9600,GT.Interfaces.Serial.SerialParity.None,GT.Interfaces.Serial.SerialStopBits.One,8);
         usbSerial.SerialLine.Open();
         usbSerial.SerialLine.DataReceived+=new
GT.Interfaces.Serial.DataReceivedEventHandler(SerialLine_DataReceived);

         GT.Timer timer = new GT.Timer(100);
         timer.Tick += new GT.Timer.TickEventHandler(timer_Tick);
         timer.Start();

         Boca = io60p16.CreatePwm(PwmPin.Pwm0, periodBoca, pulseWidthBoca,
         ModulePWM.ScaleFactor.Microseconds, false);
         Boca.Start();
         Palpebras = io60p16.CreatePwm(PwmPin.Pwm2, periodPalpebras, pulseWidthPalpebras,
  ModulePWM.ScaleFactor.Microseconds, false);
         Palpebras.Start();
      }
```

```
        void timer_Tick2(GT.Timer timer2)
        {
            countPalpebras++;
            if (abrirOlho)
            {
                changePosition(Palpebras, PwmPort, PinPalpebras, periodPalpebras, pulseWidthPalpebras);
                pulseWidthPalpebras = 1700;
                abrirOlho = false;
            }
            else if (countPalpebras >= 10 && !abrirOlho)
            {
                changePosition(Palpebras, PwmPort, PinPalpebras, periodPalpebras, pulseWidthPalpebras);
                pulseWidthPalpebras = 1100;
                abrirOlho = true;
                countPalpebras = 0;
            }
        }

        void timer_Tick(GT.Timer timer)
        {
            if (fonema)
            {
                count++;
            }
            if (fonema && (count >= 1)) //if (count == 1) //0.1 s
            {
                count = 0;
                changePosition(Boca, PwmPort, PinBoca, periodBoca, pulseWidthBoca);
                fonema = false;
            }
        }

        void changePosition(ModulePWM servo, byte port, byte pin, uint period, uint pulseWidth)
        {
            servo.SetPwm(port, pin, period * 1000, pulseWidth * 1000);
        }

        void SerialLine_DataReceived(GT.Interfaces.Serial sender, System.IO.Ports.SerialData data)
        {
            int NumberOfBytesToRead = usbSerial.SerialLine.BytesToRead;
            byte[] readInputBuffer = new byte[NumberOfBytesToRead];
            usbSerial.SerialLine.Read(readInputBuffer, 0, NumberOfBytesToRead);
            comando = new String(System.Text.UTF8Encoding.UTF8.GetChars(readInputBuffer));
            Debug.Print("Cmd: " + comando);
            pulseWidthBoca = Convert.ToUInt32(comando.Substring(0,2))*100;
            if (pulseWidthBoca <= 2600 && pulseWidthBoca >= 1400)
            {
                fonema = true;
            }
        } //End SerialLine_DataReceived
    } //End of Class Program
}
```